

Введение в язык шаблонов

Обзор языка шаблонов

Что такое шаблоны

Виды шаблонов

Что делают шаблоны

Препроцессирование и генерация исходного текста

Точки вставки

Шаблонные элементы ввода

Структура шаблона

Формат исходного текста шаблона

Файл регистра шаблонов

Настройка стандартных шаблонов

Добавление новых наборов шаблонов

Обзор языка шаблонов

Язык Шаблонов Clarion for Windows является гибким сценарным языком, дополненным управляющими структурами, элементами и т. д. Язык шаблонов “управляет” Генератором Приложений как во время разработки приложения, так и в процессе генерации исходного текста.

- Во время разработки приложения программист получает вопросы об особенностях разрабатываемого приложения. Эти информационные запросы берут начало непосредственно из шаблонов.
- Во время создания исходного текста шаблон управляет построением операторов для каждой процедуры приложения, а также управляет тем, в какие файлы будет записан сгенерированный текст.

Таким образом, шаблоны полностью управляют Генератором Приложений. Программисту это дает гибкость в создании текста, полностью отвечающего его потребностям.

Что такое шаблоны

Шаблон является законченным набором инструкций, включающим операторы языка шаблонов и целевого языка, которые Генератор Приложений использует при обработке введенных программистом параметров приложения, а затем при генерации исходного текста на целевом языке программирования (обычно, но не обязательно, на языке Clarion).

Clarion-шаблоны являются многократно используемыми. Они генерируют только тот текст, который требуется в каждом частном случае их применения; они не строят неиспользуемых методов. Они также полиморфны, т. к. программист задает свойства и функции каждого набора шаблонов, который требуется для процедуры. Это означает, что в зависимости от потребностей программиста один шаблон может создавать различную функциональность.

Некоторые наиболее важные аспекты функциональности шаблонов, поддерживаемые Языком Шаблонов, включают:

- Поддержку диалоговых элементов #PROMPT, которые принимают ввод от разработчика, сохраняя его в определяемых пользователем шаблонных переменных (символах).
- Предопределенные шаблонные переменные (встроенные символы), содержащие информацию из словаря данных и среды Clarion for Windows.

- Специализированные вводные поля `#PROMPT`, которые предоставляют программисту для выбора список возможных файлов данных или их ключей.
- Безусловные и условные управляющие структуры `#FOR`, `#LOOP`, `#IF`, `#CASE`, которые разветвляют процесс генерации исходного текста в зависимости от значения выражения или содержимого символа (переменной). Это позволяет Генератору Приложений создавать в точности тот исходный текст, который требуется для получения функциональности, которую определил программист в своем приложении.
- Операторы `#EMBED`, которые определяют те точки, где разработчик может вставлять или не вставлять свой собственный исходный текст для дальнейшей настройки приложения.
- Поддержка шаблонов текста `#CODE`, диалоговых шаблонов `#CONTROL` и распределенных шаблонов `#EXTENSION`, которые приносят свои специфические возможности в любой процедурный шаблон. Это обеспечивает свободный полиморфизм процедур в том отношении, что процедура может включать возможности, обычно присущие различным типам процедур.

Текст шаблона содержится в одном или нескольких ASCII файлах (`*.TPL` или `*.TPW`), которые Генератор Приложений предварительно компилирует и размещает в файле `REGISTRY.TRF`. Это регистр шаблонов, который используется Генератором Приложений во время разработки приложений.

Будучи занесенным в регистр, шаблон становится доступен для многократного использования в каждом новом приложении. Для каждого приложения он генерирует подходящий исходный текст в соответствии со словарем данных и особенностями, которые задал разработчик, работая с Генератором Приложений.

Программист может настроить шаблоны в регистре (или в файлах `*.TP*`) так, чтобы они соответствовали требованиям его собственных стандартов проектирования. Это означает, что каждый процедурный шаблон может быть разработан так, чтобы он возникал в начале работы в точном соответствии с требованиями программиста. Могут быть созданы многочисленные “стандартные” точки входа и программист в каждой из процедур получит возможность выбора между ними.

После настройки исходного текста шаблона (файла `*.TP*`) Генератор Приложений автоматически обновляет регистр. После настройки регистра, при необходимости, исходные тексты шаблонов могут быть восстановлены из регистра. Генератор приложений всегда делает копию загруженного в регистр шаблона при создании процедуры или при добавлении в процедуру текстовых, диалоговых или распределенных шаблонов. После того, как произведено копирование, программист настраивает процедуру, чтобы получить в точности те функции, которые должны быть у этой процедуры приложения.

Язык шаблонов может применяться не только для генерации исходных текстов, он может быть использован для создания пополняемого набора служебных программ (см. `#UTILITY`).

Виды шаблонов

Имеются четыре основных вида шаблонов: процедурный, текстовый, диалоговый и распределенный шаблоны.

- Процедурные шаблоны `#PROCEDURE` создают процедуры и/или функции приложения. Это то, что вы выбираете, когда начинаете работать с “ToDo” процедурой в Генераторе Приложений.

- Текстовые шаблоны `#CODE` генерируют выполняемый текст программы в указанной точке вставки. Разработчик может вставлять их в процедуру только в точках вставки текста. В них доступен список допустимых текстовых шаблонов из которого и производится выбор.

- Диалоговые шаблоны `#CONTROL` размещают связанный набор (из одного или нескольких) диалоговых элементов в окне процедуры и формируют исполняемый программный текст в точках вставки процедуры для обеспечения стандартного функционирования этих диалоговых элементов.

- Распределенные шаблоны `#EXTENSION` генерируют исполняемый текст в одной или более точках вставки процедуры, чтобы придать ей функции, не связанные с какими-либо диалоговыми элементами.

Что делают шаблоны

Файлы с текстами шаблонов содержат операторы языка шаблонов и обычные операторы “целевого” языка программирования, которые Генератор Приложений размещает в генерируемых файлах с исходными текстами. Они также содержат элементы ввода для построения диалогов Генератора Приложений, которые определяют типовые настройки процесса генерации текста, которые может делать разработчик.

Указания программиста (или их отсутствие) в диалоговых элементах Генератора Приложений настраивают управляющие операторы, которые обрабатывают текст на языке шаблонов, формируя логику генерации исходного текста. Шаблоны содержат также управляющие операторы, которые определяют, как Генератор Приложений должен обрабатывать стандартный текст. Функция шаблона состоит в генерации на “целевом” языке настроенного по указаниям программиста исходного текста, соответствующего структуре окна или документа.

В шаблонах могут быть строки текста, которые вставляются непосредственно в генерируемый текст. Например, если Вы сохраняете в окне приложения стандартный пункт меню с командой `Сору`, следующий текст появится в сгенерированной процедуре в точности так, как он записан в шаблоне:

```
ITEM('&Сору'),USE(?Сору),STD(STD:Сору),MSG
```

('Copy item to Windows clipboard')

Некоторые типичные тексты в шаблонах являются смесью операторов "целевого" (Clarion) языка и языка шаблонов. Например, когда в генерируемый исходный текст нужно вставить значение шаблонной переменной (символа), Генератор Приложений по мере генерации текста заменяет символ на его значение, которое будет использоваться в приложении. В тексте шаблона знак процента (%) определяет переменную (символ). В следующем примере Генератор Приложений будет подставлять метку диалогового элемента при записи в файл исходного текста.

```
SELECT(%Control)
```

Чтобы обеспечить в процессе разработки настройку первоначальных шаблонов, язык шаблонов Clarion имеет операторы, которые образуют пользовательский интерфейс шаблона, таким образом Генератор Приложений может запрашивать у разработчика необходимую информацию для настройки приложения. Основу интерфейса составляют командные кнопки, переключатели, радиокнопки и вводные поля, размещаемые в диалоге Procedure Properties. Эти операторы могут также формировать специальные диалоговые окна для приема данных от разработчика. Таким образом, при работе в Генераторе Приложений некоторые диалоги и элементы интерфейса, которые видит разработчик, не являются частью Генератора Приложений, они задаются шаблоном.

Например, следующее предложение выводит диалог выбора файла из словаря данных приложения, а затем запоминает имя указанного программистом файла данных в переменной (символе) с именем %MyFile:

```
#PROMPT ('Pick a file',FILE),%MyFile
```

Не имеет значения, как программист назовет файлы и поля и какой использует драйвер базы данных. Программист выберет их в диалоге выбора файла. Шаблон содержит также управляющие структуры для указания Генератору Приложений того, как должен создаваться текст (такие как #IF, #LOOP, #CASE). Эти управляющие операторы работают точно также, как и управляющие операторы языка Clarion.

Препроцессирование и генерация исходного текста

Перед тем, как позволить Вам создать приложение с использованием шаблонов, Генератор Приложений предварительно обрабатывает файлы (.TPL и .TPW) с текстом шаблонов. Генератор Приложений проверяет актуальность регистра, проверяя времена создания и размеры всех файлов, содержащих тексты шаблонов.

Генератор Приложений использует загруженные в регистр в двоичном виде шаблоны по мере получения от разработчика настроек в доступных с помощью диалога диалогах

и запросах к пользователю. Генератор Приложений запоминает в файле .APP начальное состояние шаблона каждой процедуры и произведенные в нем настройки.

Во время генерации текста Генератор Приложений обрабатывает не шаблон, а процедуры, как они были записаны в файл .APP. Вот некоторые важнейшие действия, которые он предпринимает для получения исходного текста:

- Он выполняет управляющие операторы языка шаблонов, чтобы в правильном порядке применить настройки шаблонов и процедур.
- Он подставляет значения символов шаблона - как встроенных, так и определяемых пользователем.
- Он создает файлы исходного текста и записывает в них строка за строкой сгенерированный шаблоном исходный текст, подставляя ранее вычисленные значения символов.
- Он изучает точки вставки и записывает исходный текст, как он был включен разработчиком и записан в .APP файл, в соответствующие места в сгенерированном исходном тексте.

Точки вставки

Одним из наиболее важных операторов языка шаблонов является #EMBED, который определяет точку вставки. Это расширяет структуру и возможности процедурного шаблона, позволяя программистам добавлять их собственный текст. Точка вставки именуется “точкой”, в которой разработчик может добавлять свой собственный исходный текст. Это также “точки”, где может размещаться исходный текст, генерируемый диалоговыми и распределенными шаблонами.

Каждый процедурный шаблон имеет определенное количество стандартных точек, в которых разрешены вставки текста. Обычно это точки, которые совпадают с сообщениями (событиями) в операционной среде (Windows), как например, когда конечный пользователь покидает или выбирает поле. Составляющий шаблон программист может дополнить или сократить этот список.

При настройке шаблона разработчиком, нажатие на кнопку Embeds в диалоге Procedure Properties дает доступ ко всем возможным точкам вставки. Выбор Actions из меню Формatera Окна тоже дает доступ к точкам вставки, но только для указанного диалогового элемента. Разработчик добавляет в точку вставки либо вручную набранный в редакторе, либо созданный текстовым шаблоном настраивающий текст. Точки вставки также являются теми точками, в которых диалоговые и распределенные шаблоны размещают текст для выполнения своих функций.

Генератор Приложений загружает текст точек вставки (независимо от его

происхождения) в .APP файл. В процессе генерации текста Генератор Приложений обрабатывает шаблон, генерируя исходный текст, при достижении точки вставки, он размещает в генерируемом тексте строка за строкой вставляемый текст разработчика.

Шаблонные элементы ввода

Операторы проверки ввода и операторы вводных полей разного типа размещают диалоговые элементы в диалогах Procedure Properties и Actions, которые видит разработчик, применяя шаблон для создания приложения. Они покрывают диапазон от простых строк, сообщающих разработчику, что делать (`#DISPLAY`), до командных кнопок, переключателей и радиокнопок. Есть также специальные типы вводных полей, которые предоставляют программисту выбор из списка возможностей, как например, поля из словаря данных. Для получения информации от программиста в окне Procedure Properties, диалоге Actions или в настраиваемом диалоге применяются стандартные диалоговые элементы Windows.

Обычные типы диалоговых элементов: вводные поля, переключатели, радиокнопки и выпадающие списки - все непосредственно поддерживается оператором `#PROMPT`.

Оператор `#PROMPT` объединяет в одном операторе приглашение, вводной диалоговый элемент и определяемый символ в одном операторе. Общий формат таков: ключевое слово `#PROMPT`, строка для вывода в качестве приглашения, тип символа, символ или имя переменной. Генератор Приложений размещает приглашение и вводной диалоговый элемент в окне Procedure Properties или диалоге Actions (в зависимости от того, происходят они из процедурного шаблона или из текстового, диалогового или распределенного шаблона). Когда разработчик заполняет вводной диалоговый элемент значением, затем закрывает диалог, символ получает и сохраняет полученное значение.

Оператор `#BUTTON` обеспечивает дополнительное “пространство” разработчику для ввода данных, когда их требуется ввести больше, чем может быть размещено в одном диалоге. Оператор размещает в диалоге кнопку, которая при нажатии выводит дополнительный настраиваемый диалог. Дополнительные диалоги называются “страницы диалога”.

Оператор `#ENABLE` позволяет условно задействовать диалоговые элементы в зависимости от содержимого в других диалоговых элементах. Оператор `#BOXED` группирует взаимосвязанные диалоговые элементы. После того, как программист ввел данные, оператор `#VALIDATE` позволяет шаблону проверить их правильность.

Эти инструменты обеспечивают большую гибкость в том, какого типа информацию шаблон должен получить у программиста. Они также обеспечивают многочисленные способы ускорения работы программиста, обеспечивая “списки для выбора” в которых программист может найти что-либо подходящее.

Взаимодействие со Словарем Данных

Для того, чтобы создать текст программы специально для продекларированной базы данных, шаблоны широко используют информацию из Словаря Данных. Имеется несколько специальных символов, которые предоставляют шаблонам доступ ко всем декларациям: %File, %Field, %Key и %Relation. Эти и все зависящие от них символы дают шаблонам доступ ко всей информации Словаря Данных.

Обратите особое внимание на символы %FileUserOptions, %FieldUserOptions, %KeyUserOptions и %RelationUserOptions. Это символы, которые содержат значения, которые пользователь вводит на страничках Options в диалогах FileProperties, FieldProperties, KeyProperties и RelationProperties. Это может быть мощным средством настройки выхода Словаря Данных.

Наилучшим способом использования этих %UserOptions символов состоит в записи в них специфических настроек в виде разделенного запятыми списка атрибутов с параметрами, которые затем анализируются шаблоном. Это придает им такой же вид, как и у атрибутов структур данных в языке Clarion. Поступая таким образом, Вы получаете возможность использовать для извлечения параметров встроенную шаблонную функцию EXTRACT. Например, пользователь ввел в поле UserOptions для поля следующее:

```
MYCUSTOMOPTION(On)
```

Код шаблона может его проанализировать используя функцию EXTRACT:

```
#IF(EXTRACT(%FieldUserOptions,'MYCUSTOMOPTION',1) = On)  
#! Выполнить нечто, соответствующее включенной опции.  
#ENDIF
```

Это очень мощный инструмент, который позволяет бесконечно гибко управлять генерацией исходного текста.

Структура шаблона

Формат исходного текста шаблона

Структура ASCII файла с исходным текстом шаблона отличается от структуры файла с исходным текстом на Clarion. При прочтении исходного ASCII текста шаблона руководствуйтесь следующими правилами:

- Любой оператор, начинающийся символом решетки (#) является оператором языка шаблонов.
- Знак процента (%) перед любым элементом в любом операторе (языка шаблонов или “целевого” языка) указывает на шаблонный символ (переменную), который обрабатывается Генератором приложений во время генерации текста.
- Любой оператор, который начинается не с символа решетки (#), является оператором “целевого” языка, который записывается непосредственно в файл исходного текста.

Текст в файлах шаблонов организован по секциям, которые ограничиваются началом следующей секции или концом файла. Текст шаблона обычно делится на десять секций.

- #TEMPLATE начинает набор шаблонов (класс шаблонов). Это первый оператор в наборе шаблонов (обязательный), который идентифицирует набор шаблонов при регистрации.
- #APPLICATION начинает секцию управления генерацией исходных текстов. Это секция шаблона, которая управляет выводом текста на “целевом” языке в готовые для компиляции выходные файлы исходного текста. Один из зарегистрированных наборов шаблонов должен иметь секцию #APPLICATION.
- #PROGRAM начинает глобальную секцию генерируемого исходного текста. Один из зарегистрированных наборов шаблонов должен иметь секцию #PROGRAM.
- #MODULE начинает секцию, которая генерирует начало модуля исходного текста, не являющегося главным файлом (файлом программы). Один из зарегистрированных наборов шаблонов должен иметь секцию #MODULE.
- #PROCEDURE начинает процедурный шаблон. Это основной шаблон, генерирующий процедуры и функции “целевого” языка.
- #GROUP начинает повторно вводимую группу операторов, содержащую текст, который может быть вставлен (#INSERTed) в любую другую секцию шаблона. Это равнозначно процедуре или функции на языке шаблонов.
- #CODE начинает секцию текстового шаблона, который генерирует исполняемый текст в указанной точке вставки. Разработчик может вставлять их только в точках вставки в процедуре. Выбор производится из появляющегося списка доступных текстовых шаблонов.
- #CONTROL начинает диалоговый шаблон. Диалоговые шаблоны размещают

связанный набор (из одного или нескольких) диалоговых элементов в окне процедуры и генерируют исполняемый текст в точках вставки процедуры, чтобы обеспечить стандартное функционирование диалоговых элементов.

- `#EXTENSION` начинает распределенный шаблон. Распределенные шаблоны генерируют исполняемый текст в одной или более точках вставки в процедуре, чтобы дополнить процедуру функциями, не связанными с каким либо диалоговым элементом.
- `#UTILITY` начинает секцию исполняемых служебных программ (утилит). Это необязательная секция шаблона, которая выполняет вспомогательные действия, такие как анализ перекрестных ссылок или генерация документации. Секция аналогична `#APPLICATION` в том, что генерирует выход в ASCII файлы.

Набор шаблонов должен иметь секцию `#TEMPLATE` для именования набора шаблонов в регистре шаблонов. По крайней мере один из зарегистрированных наборов шаблонов должен иметь секции `#APPLICATION`, `#PROGRAM` и `#MODULE`.

Файл регистра шаблонов

Файл регистра шаблонов (`REGISTRY.TRF`) является специализированным хранилищем данных, в котором запоминаются тексты шаблонов и их стандартные параметры в двоичной форме. Все доступные Генератору приложений элементы шаблонов извлекаются из регистра. По мере того, как Вы добавляете элементы шаблона в Ваше приложение, Генератор приложений извлекает их из регистра и затем добавляет его вместе с Вашими настройками в `.APP` файл.

Хранение шаблонов в двоичном регистре обеспечивает следующие преимущества.

- Быструю работу во время проектирования.
- Возможность изменять стандартные установки в регистре с помощью стандартных средств разработки приложения (таких как Форматер Окон). Например можно изменить стандартное окно процедурного шаблона без написания исходного текста шаблона.

Источником данных для `REGISTRY.TRF` служат файлы с текстами шаблонов (`.TPL` и `.TPW`), которые устанавливаются в подкаталог `TEMPLATE`. Генератор Приложений может читать и регистрировать `.TPL` файлы, добавляя их к дереву регистра шаблонов. Файлы `.TPW` обычно содержат процедурные или текстовые шаблоны, которые включаются в обработку вместе с файлом `.TPL` оператором `#INCLUDE` в файле `.TPL`. Это позволяет автору шаблона логически разделить несопоставимые компоненты шаблона.

Стандартный файл шаблонов для Clarion for Windows это `CW.TPL`. Этот файл использует оператор `#INCLUDE` для указания на другие `.TPW` файлы, которые находятся в каталоге `\CW\TEMPLATE`.

Настройка стандартных шаблонов

Есть два способа настройки шаблонов:

- Вы можете отредактировать исходный текст шаблонов в файлах .TPL и .TPW.

Всегда полезно сделать резервную копию перед внесением любых изменений в поставляемые шаблоны. При прямом редактировании исходного текста шаблонов можно изменить тип генерируемого исходного текста или используемую при генерации текста логику.

Можно заставить шаблоны генерировать текст так, как если бы он был написан вручную. Можно расширить функциональность шаблона, добавляя собственные свойства. Например, можно добавить в каждой процедуре элемент диалога, который позволит генерировать в начале каждой процедуры “блок комментариев”, содержащий указания для сопровождающего приложение программиста.

Добавление следующего текста в конец любого существующего набора шаблонов обеспечивает эту модификацию:

```
#EXTENSION (CommentBlock,'Add a comment block to the
procedure'),PROCEDURE
#PROMPT ('Comment Line',@S70),%MyComment,MULTI ('Programmer Comments')
#ATSTART
#FOR (%MyComment)
!%MyComment
#ENDFOR
#ENDAT
```

Этот текст создает распределенный шаблон, который доступен в любой процедуре приложения. При проектировании процедуры добавьте распределенный шаблон CommentBlock, затем добавляйте комментарии в строке Comment Line каждый раз при модификации процедуры. Во время генерации текста каждая строка комментария будет появляться в сгенерированном тексте вместе с предшествующим ей восклицательным знаком. Блок комментариев появится в тексте непосредственно перед оператором PROCEDURE или FUNCTION.

Если Вы хотите, чтобы этот распределенный шаблон применялся во всех процедурах, нужно войти в регистр шаблонов и добавить этот распределенный шаблон ко всем стандартным процедурам каждого процедурного шаблона. Таким образом, можно быть уверенным, что он будет всегда использован и можно даже разместить этот диалог в окне Procedure Properties, отметив переключатель Show on Properties во время добавления распределенного шаблона к процедурным шаблонам.

Сделав изменения, нужно выбирать в меню Setup-Template Registry, либо открыть существующее приложение, либо создать новое приложение. Необходимо убедиться, что переключатель Re-registry When Changed в диалоге Registry Options включен (отмечен). Генератор Приложений автоматически препроцессирует шаблоны, чтобы обновить регистр после Ваших изменений в файлах исходных текстов шаблонов.

- Используя регистр шаблонов можно добавить или изменить элементы стандартного пользовательского интерфейса процедурного шаблона, такие как устройство стандартного окна или компоновка отчета или стандартные глобальные и локальные переменные.

Подсветка процедурного шаблона в регистре шаблонов и нажатие на кнопку Properties открывает диалог Procedure Properties без каких либо элементов настраивающего диалога, которые обычно видны во время разработки приложения.

Любая действующая кнопка диалога может использоваться для создания стандартной точки входа в процедуру. Можно установить точку входа гораздо ближе к законченной процедуре, что потребует меньшего количества настроек во время разработки приложения.

Если процедура позволяет, можно использовать формтеры окон и отчетов или определить дополнительные данные, нажимая соответствующие кнопки. После настройки регистра шаблонов, настройки можно экспортировать в файлы с исходными текстами шаблонов. Это полезно для совместного использования одних и тех же настроек группой разработчиков.

Чтобы дополнить исходные тексты шаблонов изменениями, сделанными в регистре шаблонов, нужно нажать в диалоге Template Properties кнопку Regenerate. Это дополнит файлы .TPL и .TPW сделанными изменениями.

Добавление новых наборов шаблонов

Добавление другого набора шаблонов либо от независимого поставщика, либо собственных, представляет очень простой процесс. Есть только одно требование к новому набору шаблонов - оператор #TEMPLATE для идентификации набора шаблонов в регистре. Конечно, он должен содержать новые специфические процедурные, текстовые, диалоговые и распределенные шаблоны для добавления в регистр.

Например, следующий текст является абсолютно правильным набором шаблонов без каких-либо дополнений:

```
#TEMPLATE (PersonalAddOns,'My personal Template set')
#CODE (ChangeProperty,'Change control property')
#PROMPT ('Control to change',CONTROL),%MyField,REQ
```

```
#PROMPT ('Property to change',@S20),%MyProperty,REQ  
#PROMPT ('New Value',@S20),%MyValue,REQ  
%MyField{%MyProperty} = '%MyValue'  
#<!Change the %MyProperty of %MyField
```

После регистрации этот набор шаблонов появится в регистре шаблонов как класс PersonalAddOns, содержащий только текстовый шаблон ChangeProperty.

После регистрации набора шаблонов в регистре, все его компоненты становятся доступными программисту для разработки приложений, также как и все остальные компоненты всех зарегистрированных наборов. Это дает программисту гибкость в “смешивании и подборе” их компонент во время разработки.

Например, программист мог бы создать процедуру из процедурного шаблона стандартного набора шаблонов Clarion, разместить в ней диалоговый шаблон независимого поставщика, добавить в точку вставки текстовый шаблон от другого независимого поставщика, а затем добавить распределенный шаблон из лично написанного набора шаблонов. Во время генерации текста все эти разнообразные компоненты собираются вместе для создания полностью функциональной процедуры, которая решает все требуемые программистом задачи (и ничего более). Это и составляет истинную силу шаблонно-ориентированного программирования в среде Clarion.

Устройство шаблона

Секции шаблона

#TEMPLATE (начало набора шаблонов)
#APPLICATION (секция управления генерацией текста)
#PROGRAM (область глобалов)
#MODULE (область модуля)
#PROCEDURE (начало процедурного шаблона)
#GROUP (повторно используемая группа операторов)
#UTILITY (секция утилит)
#CODE (определение текстового шаблона)
#CONTROL (определение диалогового шаблона)
#EXTENSION (определение распределенного шаблона)

Точки вставки

#EMBED (задание точек вставки исходного текста)
#AT (вставка текста в точку вставки)
#ATSTART (инициализация шаблона)
#ATEND (завершение шаблона)
Поддержка целостности шаблона
#WHERE (указание доступности #CODE в точке вставки)
#RESTRICT (ограничения на использование секции)
#ACCEPT (разрешение на использование секции)
#REJECT (запрет на использование секции)

Секции шаблона

#TEMPLATE (начало набора шаблонов)

#TEMPLATE (*name, description*)

#TEMPLATE	Отмечает начало набора шаблонов.
<i>name</i>	Название набора шаблонов, которое идентифицирует их в регистре и операторах языка шаблонов. Должно быть допустимым именем языка Clarion.
<i>description</i>	Строковая константа, описывающая набор шаблонов для регистра шаблонов и Генератора Приложений.

Оператор #TEMPLATE отмечает начало набора шаблонов. Он должен быть первым отличным от комментария оператором в файле шаблонов.

Регистр шаблонов позволяет зарегистрировать для Генератора Приложений многочисленные наборы шаблонов. Каждая секция шаблона (#APPLICATION, #PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION и #GROUP) в шаблоне единственным образом идентифицируется именем в операторе #TEMPLATE и именем секции. Это позволяет различным наборам шаблонов иметь одинаковые имена секций без каких-либо противоречий и позволяет программисту попеременно использовать для генерации приложения шаблоны из разных наборов, которые были получены из разных источников.

Пример:

```
#TEMPLATE (SampleTemplate,'This is a sample Template')
#include ('FileTwo.TPX')
#include ('FileThree.TPX')
```

#APPLICATION (секция управления генерацией текста)

#APPLICATION (*description*) [, **HLP** (*helpid*)]

#APPLICATION	Отмечает начало секции управления генерацией текста
<i>description</i>	Строковая константа, описывающая секцию приложения
HLP	Указывает на доступность справки.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо "строка контекста".

Оператор #APPLICATION отмечает начало секции управления генерацией текста. Секция заканчивается оператором начала следующей секции (#PROGRAM, #MODULE,

Любой определяемый символ, объявленный в секции #APPLICATION, доступен для использования в любой секции генерируемых шаблонов. Любой элемент диалога в этой секции размещается в окне Global Properties и имеет глобальную область видимости.

[illegible]

Files ..)

```

    #SET (%AccessMode,'22h')           #! default access 'open'
#ENDIF                                #! END (IF Shared Files ...)
#IF (%INIFile = 'Program Name.INI')   #! IF using program.ini
    #SET (%INIFileName, %Application & '.INI') #! SET the file name
#    E    L    S    E
    #! ELSE (IF NOT using Program.ini)
    #SET (%INIFileName,%INIFileName)     #! SET the file name
#ENDIF                                    #! END (IF using program.ini)
#!----- Main Source Code Generation Loop.
#DECLARE (%GlobalRegenerate)           #! Flag that controls generation
#IF (~%ConditionalGenerate OR %DictionaryChanged OR %RegistryChanged)
    #SET (%GlobalRegenerate,%True)      #! Generate Everything
#ELSE                                   #! ELSE (If no global change)
    #SET (%GlobalRegenerate,%False)     #! Generate changed modules only
#ENDIF                                  #! END (IF Global Change)
#SET (%BuildFile,(%Application & '.TMS')) #! Make temp program filename
#FOR (%Module), WHERE (%Module <> %Program) #! For all member
modules
    #MESSAGE('Generating Module: ' & %Module, 1) #! Post generation message
    #IF (%ModuleChanged OR %GlobalRegenerate) #! IF module to be
generated
        #FREE(%ModuleProcs)             #! Clear module prototypes
        #FREE(%ModuleFilesUsed)         #! Clear files used
        #CREATE(%BuildFile)             #! Create temp module file
        #FOR (%ModuleProcedure)         #! FOR all procs in module
            #FIX (%Procedure,%ModuleProcedure) #! Fix current procedure
        #MESSAGE('Generating Procedure: ' & %Procedure, 2) #! Post generation mes-
sage
        #GENERATE (%Procedure)           #! Generate procedure code
    #ENDFOR                             #! END (For all procs in
module)
        #CLOSE(%BuildFile)              #! Close last temp file
        #CREATE(%Module)                #! Create a module file
        #GENERATE (%Module)             #! Generate module header
        #APPEND(%BuildFile)             #! Append the temp mod file
        #CLOSE(%Module)                #! Close the module fileENDIF
    #! END (If module to be...)
#ENDFOR                                #! END (For all member modules)
#FIX (%Module,%Program)                #! FIX to program module
#MESSAGE('Generating Module: ' & %Module, 1) #! Post generation message
#FREE(%ModuleProcs)                   #! Clear module prototypes
#FREE(%ModuleFilesUsed)               #! Clear files used

```

```

#CREATE(%BuildFile)           #! Create temp module file
#FOR (%ModuleProcedure)       #! For all procs in module
    #FIX (%Procedure,%ModuleProcedure)    #! Fix current procedure
    #MESSAGE('Generating Procedure: ' & %Procedure, 2) #! Post generation message
    #GENERATE (%Procedure)     #! Generate procedure code
#ENDFOR                       #! EndFor all procs in module
#CLOSE()                      #! Close last temp file
См. также #GENERATE

```

#PROGRAM (область глобалов)

```
#PROGRAM ( name, description [, target, extension ] ) [, HLP ( helpid ) ]
```

#PROGRAM	Отмечает начало главного модуля программы.
<i>name</i>	Название секции #PROGRAM, которое идентифицирует ее в регистре и операторах языка шаблонов. Должно быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая #PROGRAM для регистра шаблонов и Генератора Приложений.
<i>target</i>	Строковая константа, описывающая для какого целевого языка составлен шаблон. Если опущена, то принимается Clarion.
<i>extension</i>	Строковая константа, указывающая расширение имени файла для записи сгенерированного текста. Если опущено, используется .CLW.
HLP	Указывает на доступность справки.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо "строка контекста".

Оператор #PROGRAM отмечает начало шаблона главного модуля программы. Секция #PROGRAM заканчивается оператором начала следующей секции (#MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. Только одна секция #PROGRAM может быть в наборе шаблонов.

Операторы #BUTTON, #PROMPT и #DISPLAY недопустимы в секции #PROGRAM. Глобальные элементы диалога располагаются в секции #APPLICATION.

Пример:

```

#PROGRAM (CLARION,'Standard Clarion Shipping Template')
PROGRAM                                !PROGRAM statement required
INCLUDE('Keycodes.clw')
INCLUDE('Errors.clw')
INCLUDE('Equates.clw')

```

#MODULE (область модуля)

```
#MODULE ( name, description [, target, extension ] ) [, HLP( helpid ) ][, EX-  
TERNAL ]
```

#MODULE	Отмечает начало секции модуля.
<i>name</i>	Имя модуля, которое идентифицирует его в регистре и операторах языка шаблонов. Должно быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая секцию #MODULE для регистра шаблонов и Генератора Приложений.
<i>target</i>	Строковая константа, описывающая, для какого целевого языка составлен шаблон. Слово EXTERNAL применяется для указания на внешний исходный или объектный модуль. Если опущено, то принимается Clarion.
<i>extension</i>	Строковая константа, указывающая расширение имени файла для записи сгенерированного текста. Если опущено, используется .CLW.
HLP	Указывает на доступность справки.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо “строка контекста”.
EXTERNAL	Указывает, что не будет генерироваться никакого текста.

Оператор #MODULE определяет начало секции шаблона, которая размещает текст в области данных каждого генерируемого исходного модуля. Секция #MODULE заканчивается оператором начала следующей секции (#PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. В наборе шаблонов может быть несколько операторов #MODULE.

Сгенерированный секцией #MODULE текст (обычно) размещается в начале файла исходного текста, сгенерированного Генератором Приложений.

Операторы #BUTTON, #PROMPT, и #DISPLAY недопустимы в секции #MODULE.

Пример:

```
#MODULE (ExternalOBJ,'External .OBJ module','EXTERNAL','OBJ'),EXTERNAL
#MODULE (ExternalLIB,'External .LIB module','EXTERNAL','LIB'),EXTERNAL
#MODULE (GENERATED,'Clarion MEMBER module')
    MEMBER('%Program')      !MEMBER statement is required
%ModuleData                #!Data declarations local to the Module
```

#PROCEDURE (начало процедурного шаблона)

```
#PROCEDURE ( name, description [, target ] )[, REPORT ][, WINDOW ]
```

[, HLP (helpid)][, PRIMARY (message [, flag])][, QUICK(wizard)]

#PROCEDURE*name*

Отмечает начало шаблона процедуры.

Идентификатор шаблона процедуры. Должен быть допустимым в языке Clarion именем.

description

Строковая константа, описывающая шаблон процедуры.

target

Строковая константа, описывающая, для какого целевого языка составлен шаблон. Если опущено, то принимается Clarion.

REPORT

Позволяет использовать форматер документов (отчетов).

WINDOW

Позволяет использовать форматер окон.

HLP

Указывает на доступность справки.

helpid

Строковая константа содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо “строка контекста”.

PRIMARY

Указывает, что по крайней мере один файл должен быть показан в схеме файлов процедуры.

QUICK

Указывает, что для процедуры имеется Мастер-утилита (#UTILITY), которая выполняется при установке режима Use Procedure Wizard.

wizard

Идентификатор (включающий при необходимости указание класса) шаблона Мастер-утилиты.

Оператор отмечает начало процедурного шаблона. Процедурный шаблон содержит операторы языка шаблонов и целевого языка, которые используются для генерации исходного текста процедуры в приложении. Секция #PROCEDURE заканчивается оператором начала следующей секции (#PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. В наборе шаблонов может быть несколько операторов #PROCEDURE, но у них должны быть уникальные идентификаторы.

Пример:

```
#PROCEDURE (ProcName1,'This is a sample procedure')
#PROCEDURE (ProcName2,'This is a sample window procedure'),WINDOW
#PROCEDURE (ProcName3,'This is a sample report procedure'),REPORT
#PROCEDURE (ProcName4,'This is a sample anything
procedure'),WINDOW,REPORT
```

#GROUP (повторно используемая группа операторов)

#GROUP (symbol [, [type] parameters]) [, AUTO] [, HLP (helpid)]

#GROUP

Начинает секцию шаблона, которая может вставляться в разных местах шаблона.

<i>symbol</i>	Определяемый символ, используемый как идентификатор #GROUP.
<i>type</i>	Тип передаваемого параметра: LONG, REAL, STRING или * (звездочка). Звездочка (*) указывает, что это параметр-переменная (передается по адресу), значение которого может быть изменено в #GROUP. LONG, REAL и STRING указывают, что это параметр-значение (передается по значению), значение которого не может быть изменено в #GROUP. Если type опущен, parameter передается как STRING.
<i>parameters</i>	Определяемые символы, с помощью которых ссылаются на передаваемые в #GROUP значения. В #GROUP может быть передано несколько параметров, разделенных запятыми. Все указанные параметры должны передаваться в #GROUP; они не могут опускаться.
AUTO	Открывает новую область видимости для группы. Это означает, что любой оператор #DECLARE внутри #GROUP не будет доступен в генерируемой #PROCEDURE. Передача параметров в #GROUP неявно открывает новую область видимости.
HLP <i>helpid</i>	Указывает на доступность справки. Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо “строка контекста”.

#GROUP определяет начало текстовой секции, которая генерируется в исходный текст. Секция #GROUP может содержать операторы языка шаблонов и целевого языка. Секция #GROUP заканчивается оператором начала следующей секции (#PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. В одном наборе шаблонов различные #GROUP не могут быть определены с одинаковыми symbol.

Передаваемые в #GROUP параметры распадаются на две категории: параметры-значения и параметры-переменные.

Параметры-значения объявляются как определяемые символы с необязательным type и передаются “по значению” (передается копия значения). Любые символы или выражения могут передаваться как параметр-значение. Когда многозначный символ передается как параметр-значение, передается только его текущий экземпляр.

Параметры-переменные объявляются как определяемые символы с предшествующей звездочкой (*) (и без type). Параметры-переменные передаются “по адресу” и любое изменение их значений в #GROUP изменяет значение переданного символа. Только символы могут передаваться в #GROUP в качестве параметра-переменной. Когда многозначный символ передается как параметр-переменная, передаются все его экземпляры.

Входящие в #GROUP операторы генерируются оператором #INSERT. #GROUP может содержать операторы #EMBED, определяющие точки вставки исходных текстов.

Пример:

```
#GROUP (%GenerateFormulas) #!A #GROUP without parameters
  #FOR (%Formula)
    #IF (%FormulaComputation)
      %Formula = %FormulaComputation
    #ELSE
      IF(%FormulaCondition)
        %Formula = %FormulaTrue
      ELSE
        %Formula = %FormulaFalse
      END
    #ENDIF
  #ENDFOR
#GROUP (%ChangeProperty,%MyField,%Property,%Value)
      #!A #GROUP that receives parameters
%MyField{%Property} = '%Value' #<!Change the %Property of %MyField

#GROUP (%SomeGroup, * %VarParm, LONG %ValParm)
  #!A #GROUP that receives a variable-parameter and a value-parameter
См. также:#INSERT
```

#UTILITY (секция утилит)

#UTILITY (*name*, *description*) [, **HLP** (*helpid*)] [, **WIZARD**(*procedure*)]

#UTILITY	Начинает управление генерацией утилит.
<i>name</i>	Идентификатор #UTILITY, который идентифицирует секцию в регистре шаблонов. Должен быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая секцию утилиты.
HLP	Указывает на доступность справки.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо “строка контекста”.
QUICK	Указывает, что для процедуры имеется Мастер-утилита (#UTILITY), которая выполняется при установке режима Use Procedure Wizard.
<i>wizard</i>	Идентификатор (включающий при необходимости указание класса) шаблона Мастер-утилиты.

Оператор `#UTILITY` отмечает начало секции управления выполнением утилиты. Секция `#UTILITY` заканчивается оператором начала следующей секции (`#PROGRAM`, `#MODULE`, `#PROCEDURE`, `#CONTROL`, `#CODE`, `#EXTENSION`, `#UTILITY` или `#GROUP`) или концом файла. Содержащиеся в этой секции операторы языка шаблонов управляют процессом исполнения утилиты. В наборе шаблонов может быть несколько операторов `#UTILITY`.

Секция `#UTILITY` очень похожа на секцию `#APPLICATION` в том отношении, что она позволяет иметь вывод из приложения. Назначение `#UTILITY` в том, чтобы обеспечить расширяемые прикладные утилиты для таких вещей, как программная документация или дерево вызовов процедур. Список зарегистрированных утилит выводится в меню `Utility` в среде `Clarion for Windows`.

`#UTILITY` с атрибутом `WIZARD` указывает, что она содержит `#SHEET` со страничками `#TAB`, показывает их по одной страничке, и проводит пользователя по диалогам настройки.

Пример:

```
#UTILITY (ProcCallTree, 'Output procedure call tree')
  #CREATE(%Application & '.TRE')
  Procedure Call Tree: for %Application
  -----
  #INSERT (%DisplayTree, %FirstProcedure, ' ', ' ')
  #CLOSE
  #!*****
  #GROUP (%DisplayTree, %ThisProc, %Level, %NextIndent)
    #FIX (%Procedure, %ThisProc)
    %Level+-%ThisProc (%ProcedureTemplate)
    #FOR (%ProcedureCalled)
      #IF (INSTANCE (%ProcedureCalled) = ITEMS(%ProcedureCalled))
      #INSERT (%DisplayTree, %ProcedureCalled, %Level & %NextIndent, ' ')
      #ELSE
      #INSERT (%DisplayTree, %ProcedureCalled, %Level & %NextIndent, '| ')

    #ENDIF
  #ENDFOR
```

#CODE (определение текстового шаблона)

```
#CODE ( name, description [, target ] )[, SINGLE][, HLP ( helpid )]
      [, PRIMARY ( message [, flag ] )] [, REQ( addition [, | BEFORE| ] )]
      [, | FIRST | ] [, DESCRIPTION( expression )] [, ROUTINE ]
```


| LAST |

#CODE	Отмечает начало текстового шаблона, который генерирует текст в точке вставки исходного текста.
<i>name</i>	Идентификатор текстового шаблона. Должен быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая текстовый шаблон. Общее количество знаков в операторе #CODE должно быть меньше 255. Следовательно description не должно быть слишком длинным, чтобы оператор #CODE в целом не превосходил этот предел.
<i>target</i>	Строковая константа, описывающая для какого целевого языка составлен шаблон. Если опущено, то принимается Clarion. Это ограничивает использование #CODE указанным целевым языком.
SINGLE	Указывает, что #CODE может быть только однократно использован в данной процедуре (или программе, если точка вставки находится в глобальной области).
HLP	Указывает на доступность справки.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо “строка контекста”.
REQ	Указывает, что для использования #CODE должны быть предварительно размещены #CODE, #CONTROL или #EXTENSION. Это также означает доступность всех элементов диалога и переменных требуемых шаблонов.
<i>addition</i>	Идентификатор ранее размещенного #CODE, #CONTROL или #EXTENSION из любого набора шаблонов.
BEFORE	Указывает, что текст генерируется перед тем, как генерируется текст addition.
AFTER	Указывает, что текст генерируется после генерации текста addition.
FIRST	Указывает, что текст генерируется в начале точки вставки, перед любым другим текстом.
LAST	Указывает, что текст генерируется в конце точки вставки, после любого другого текста.
DESCRIPTION	Определяет выводимое в диалоге описание шаблона #CODE, которое может многократно использоваться в данном приложении или процедуре.
<i>expression</i>	Строковая константа или выражение, содержащее выводимое описание.
ROUTINE	Указывает, что в генерируемом коде не должен выполняться автоматический отступ с первой колонки.
PRIMARY	Указывает, что для текстового шаблона должен быть указан по крайней мере один файл в схеме файлов процедуры.
<i>message</i>	Строковая константа, содержащая сообщение, которое появится в схеме файлов у главного файла #CODE.

flag Либо OPTIONAL (файл необязателен), либо OPTKEY (ключ необязателен), либо NOKEY (файл может не иметь ключей).

Оператор #CODE задает начало секции текстового шаблона, который может генерировать текст в точках вставки исходного текста. Секция #CODE заканчивается оператором начала следующей секции (#PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. В одном наборе шаблонов различные секции #CODE не могут быть определены с одинаковыми name.

#CODE генерирует свой текст в #EMBED точках вставки исходного текста. Секция #CODE может содержать операторы #PROMPT для ввода значений, необходимых для правильной генерации исходного текста. Она также может содержать операторы #EMBED, которые активизируются только после применения #CODE.

Вы можете использовать оператор #WHERE, чтобы ограничить доступность #CODE только теми точками вставки, где его применение возможно. #CODE может содержать многочисленные операторы #WHERE, чтобы точно определить все допустимые точки вставки в которых может появляться шаблон. #RESTRICT тоже позволяет ограничить доступность #CODE, основываясь на значениях выражений или операторов языка шаблонов.

Структуры #AT / #ENDAT позволяют с помощью одного #CODE генерировать текст в многочисленных точках вставки, чтобы обеспечить необходимые действия.

Пример:

```
#CODE (ChangeProperty,'Change control property')
  #WHERE (%SetupWindow-%CustomRoutines) #!Appropriate only after window
open
  #PROMPT ('Control to change',WINDOWCONTROL),%MyField,REQ
  #PROMPT ('Property to change',@S20),%Property,REQ
  #PROMPT ('New Value',@S20),%Value,REQ
  %MyField{%Property} = '%Value' #<!Change the %Property of %MyField
См. также: #EMBED,#WHERE,#RESTRICT,#AT
```

#CONTROL (определение диалогового шаблона)

```
#CONTROL ( name, description ) [, MULTI ] [, PRIMARY ( message [, flag ]
) ]
  [, WINDOW ] [, REPORT]
  [, REQ( addition [, | BEFORE      | ] ) [, | FIRST      | ]
      | AFTER      |      | LAST|
  [, DESCRIPTION( expresion ) ] ]
CONTROLS
  control statements [, #REQ ]
END
```

#CONTROL	Отмечает начало диалогового шаблона, который размещает в окне набор диалоговых элементов и генерирует в точках вставки исходного текста необходимый для управления ими исходный текст.
<i>name</i>	Идентификатор диалогового шаблона. Должен быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая диалоговый шаблон. Общее количество знаков в операторе #CONTROL должно быть меньше 255. Следовательно description не должно быть слишком длинным, чтобы оператор #CONTROL в целом не превосходил этот предел.
MULTI	Указывает, что #CONTROL может использоваться несколько раз в одном и том же окне.
PRIMARY	Указывает, что для набора диалоговых элементов должен быть указан по крайней мере один файл в схеме файлов процедуры.
<i>message</i>	Строковая константа, содержащая сообщение, которое появится в схеме файлов у главного файла #CONTROL.
<i>flag</i>	Либо OPTIONAL (файл необязателен), либо OPTKEY (ключ необязателен), либо NOKEY (файл может не иметь ключей).
WINDOW	Указывает Генератору приложений, что #CONTROL должен быть доступен в формате окна. Это действует по умолчанию, если опущены и WINDOW и REPORT.
REPORT	Указывает Генератору приложений, что #CONTROL должен быть доступен в формате документа. Если опущено, #CONTROL не может быть размещен в документе.
REQ	Указывает, что #CONTROL требует для своего применения ранее размещенных #CODE, #CONTROL или #EXTENSION.
<i>addition</i>	Идентификатор ранее размещенного #CODE, #CONTROL или #EXTENSION.
BEFORE	Указывает, что текст генерируется перед тем, как генерируется текст addition.
AFTER	Указывает, что текст генерируется после генерации текста addition.
FIRST	Указывает, что текст генерируется в начале точки вставки, перед любым другим текстом.
LAST	Указывает, что текст генерируется в конце точки вставки, после любого другого текста.
DESCRIPTION	Определяет выводимое в диалоге описание шаблона #CONTROL, который может быть многократно использован в данном приложении или процедуре.
<i>expression</i>	Строковая константа или выражение, содержащее выводимое описание.
CONTROLS	Объявляет диалоговые элементы для #CONTROL; должен завершаться оператором END. Это “псевдо-Clarion ключевое слово”


```

Key',@s20),%ChangeHotKey,DEFAULT('CtrlEnter')
#ENDENABLE
#PROMPT('Allow Deletes',CHECK),%DeleteAllowed,DEFAULT(1)
#ENABLE(%DeleteAllowed)
    #PROMPT('Delete Hot Key',@s20),%DeleteHotKey,DEFAULT('DeleteKey')
#ENDENABLE
#PROMPT('Update Procedure',PROCEDURE),%UpdateProc

CONTROLS
    LIST,AT(,270,99),USE(?List),IMM,FROM (Queue:Browse),#REQ
    BUTTON('Insert'),AT(,40,15),KEY(%InsertHotKey),USE(?Insert),MSG('Add
record')
    BUTTON('Change'),AT(,40,15),KEY(%ChangeHotKey),USE(?Change),DEFAULT,|
        MSG('Change rec')
    BUTTON('Delete'),AT(,40,15),KEY(%DeleteHotKey),USE(?Delete),MSG('Delete
record')
END
#!
    #AT(%ControlEvent),WHERE(%ControlOriginal='?Insert'    AND
%ControlEvent='Accepted')
    #IF(%InsertAllowed)
        Action = AddRecord
        %UpdateProc
    #ENDIF
#ENDAT
#!
    #AT(%ControlEvent),WHERE(%ControlOriginal='?Change'    AND
%ControlEvent='Accepted')
    #IF(%ChangeAllowed)
        Action = ChangeRecord
        %UpdateProc
    #ENDIF
#ENDAT
#!
    #AT(%ControlEvent),WHERE(%ControlOriginal='?Delete'    AND
%ControlEvent='Accepted')
    #IF(%DeleteAllowed)

        Action = DeleteRecord
        %UpdateProc
    #ENDIF

```

#ENDAT

См. также: #EMBED, #WHERE, #RESTRICT, #AT

#EXTENSION (определение распределенного шаблона)

```
#EXTENSION ( name, description [, target ] ) [, MULTI ] [, APPLICATION ]
                                     | PROCEDURE |
[ , REQ( addition [, | BEFORE | ] ) [, FIRST | ]
                                     | AFTER | | LAST |
[ , DESCRIPTION( expression ) ] [, PRIMARY ( message [, flag ] ) ]
```

#EXTENSION	Отмечает начало распределенного шаблона, который генерирует в точках вставки исходного текста необходимый исходный текст, который не привязан к какому-либо диалоговому элементу.
<i>name</i>	Идентификатор распределенного шаблона. Должен быть допустимым в языке Clarion именем.
<i>description</i>	Строковая константа, описывающая распределенный шаблон.
<i>target</i>	Строковая константа, описывающая для какого целевого языка составлен шаблон. Если опущена, то принимается Clarion.
MULTI	Указывает, что #EXTENSION может многократно использоваться в приложении или процедуре.
APPLICATION	Указывает Генератору Приложений, что #EXTENSION должен быть доступен только на глобальном уровне.
PROCEDURE	Указывает Генератору Приложений, что #EXTENSION должен быть доступен только на локальном уровне.
REQ	Указывает, что #EXTENSION требует для своего применения ранее размещенных #CODE, #CONTROL или #EXTENSION.
<i>addition</i>	Идентификатор ранее размещенного #CODE, #CONTROL или #EXTENSION.
BEFORE	Указывает, что текст генерируется перед тем, как генерируется текст <i>addition</i> .
AFTER	Указывает, что текст генерируется после генерации текста <i>addition</i> .
FIRST	Указывает, что текст генерируется в начале точки вставки, перед любым другим текстом.
LAST	Указывает, что текст генерируется в конце точки вставки, после любого другого текста.
DESCRIPTION	Определяет выводимое в диалоге описание шаблона #EXTENSION, который может многократно использоваться в данном приложении или процедуре.
<i>expression</i>	Строковая константа или выражение, содержащее выводимое описание.
PRIMARY	Указывает, что для распределенного шаблона должен быть указан по крайней мере один файл в схеме файлов процедуры.

message Строковая константа, содержащая сообщение, которое появится в
схеме файлов у главного файла шаблона #EXTENSION.
flag Либо OPTIONAL (файл необязателен), либо OPTKEY (ключ
необязателен), либо NOKEY (файл может не иметь ключей).

#EXTENSION определяет начало распределенного шаблона, который генерирует исходный текст для выполнения некоторых функций, которые не связаны непосредственно с каким-либо диалоговым элементом. Секция #EXTENSION может содержать операторы языка шаблонов и/или целевого языка. Секция #EXTENSION заканчивается оператором начала следующей секции (#PROGRAM, #MODULE, #PROCEDURE, #CONTROL, #CODE, #EXTENSION или #GROUP) или концом файла. В одном наборе шаблонов различные секции #EXTENSION не могут быть определены с одинаковыми именами.

#EXTENSION может только генерировать текст в #EMBED точках вставки исходного текста, используя структуры #AT/#ENDAT. Секция #EXTENSION может содержать операторы #PROMPT для ввода значений, необходимых для правильной генерации исходного текста. Эти элементы ввода возникают при настройке #EXTENSION с помощью кнопки Extensions. Шаблон также может содержать операторы #EMBED, которые активизируются только после применения #EXTENSION.

#RESTRICT позволяет ограничить доступность #EXTENSION, основываясь на значениях выражений или операторов языка шаблонов.

Пример:

```
#EXTENSION(Security,'Add password'),PROCEDURE
  #PROMPT('Password File',FILE),%PasswordFile,REQ
  #PROMPT('Password Key',KEY,%PasswordFile),%PasswordFileKey,REQ
                                     #PROMPT('Password
Field',COMPONENT,%PasswordFileKey),%PasswordFileKeyField,REQ
  #AT(%DataSection)
  LocalPswd  STRING(10)
  SecurityWin WINDOW
ENTRY(@s10),USE(LocalPswd),REQ
      BUTTON('Cancel'),KEY (EscKey),USE(?Cancel)
      END
  #ENDAT
  #AT(%ProcedureSetup)
OPEN(SecurityWin)
ACCEPT
  CASE ACCEPTED()
  OF ?LocalPswd
    %PasswordFileKeyField = LocalPswd
    GET(%PasswordFile,%PasswordFileKey)
```

```

    IF NOT ERRORCODE()
        LocalPswd = 'OK'
    END
    BREAK
OF ?Cancel
    BREAK
END
CLOSE(SecurityWin)
IF LocalPswd <> 'OK' THEN RETURN.
#ENDAT
См также: #EMBED, #WHERE, #RESTRICT, #AT

```

Точки вставки

#EMBED (задание точек вставки исходного текста)

```

#EMBED ( identifier, descriptor ) [, symbol ] [, HLP ( helpid )] [, DATA ] [, HIDE
]
           [, WHERE ( expression ) ] [, MAP( symbol, description ) ] [, LABEL ]

```

#EMBED	Отмечает место в шаблоне, где программист имеет возможность разместить свой собственный исходный текст.
<i>identifier</i>	Определяемый шаблонный символ, который идентифицирует точку вставки для Генератора Приложений.
<i>descriptor</i>	Строковая константа, содержащая описание положения точки вставки в шаблоне. Эта строка выводится в списке доступных окон вставки исходного текста в процедурном шаблоне.
<i>symbol</i>	Встроенный многозначный шаблонный символ. Вы можете указать несколько символов в одном операторе #EMBED.
HLP	Указывает на доступность справки для #EMBED.
<i>helpid</i>	Строковая константа, содержащая идентификатор для обращения к справочной системе. Это может быть либо ключевое слово справки, либо "строка контекста".
DATA	Указывает, что точка вставки находится в разделе данных, и что в текстовом редакторе могут использоваться форматы окна и документа.
HIDE	Указывает, что точка вставки текста не должна появляться в дереве доступных точек вставки исходных текстов. Следовательно, #EMBED будет доступен для генерации текстов только в #CODE, #CONTROL или #EXTENSION.
WHERE	Указывает, что #EMBED доступен только для тех экземпляров symbol для которых expression истинно.
<i>expression</i>	Выражение, задающее условие доступности #EMBED.

MAP	Сопоставляет <code>description</code> и <code>symbol</code> для вывода в дереве доступных точек вставки. Можно иметь столько атрибутов <code>MAP</code> , сколько
<i>description</i>	указано символов. Выражение, задающее текст для вывода в дереве доступных точек вставки.
LABEL	Указывает, что точка вставки размещена начиная с первой колонки (1), что позволяет Текстовому Редактору правильно раскрасить метки, показывая программисту, что они будут сгенерированы начиная с первой колонки (1).

`#EMBED` определяет то место в шаблоне, где программист имеет возможность вызвать процедуру, сгенерировать текст с помощью текстового шаблона или разместить свой собственный исходный текст в процедуре или функции. Генератор приложений запрашивает программиста о вызываемой процедуре или об используемом текстовом шаблоне или вызывает текстовый редактор для составления вставляемого исходного текста. `#EMBED` также используется для размещения текстов, генерируемых в секциях `#CODE`, `#CONTROL` и `#EXTENSION`. Если в точке вставки нет написанного программистом или сгенерированного шаблоном текста, то ничего не генерируется.

В секции `#PROCEDURE` исходный текст помещается в точности в ту колонку, в которой размещен `#EMBED` в шаблоне. Если `#EMBED` размещен в разделе данных `#PROGRAM`, `#MODULE` или `#PROCEDURE`, то он должен начинаться с первой (1) колонки файла шаблона, чтобы вставляемый текст мог содержать метки данных. Если оператор `#EMBED` содержит атрибут `DATA`, то в текстовом редакторе будут доступны для использования форматы окна и документа. В разделе исполняемого текста `#EMBED` может быть помещен в первую колонку, но это необязательно.

`#EMBED` может находиться в секции `#GROUP`, но этим следует пользоваться осторожно. Так как `#GROUP` может быть рекурсивной (вызывать себя), можно создать текстовую вставку, которая будет повторена для каждой итерации рекурсивно сгенерированного текста. Исходный текст генерируется начиная с той же относительной колонки, что и текст, генерируемый `#GROUP`.

Атрибут `symbol` применяется в `#EMBED` в сочетании с оператором `#FOR`, чтобы организовать разные точки вставки для каждого из экземпляров многозначного символа. `#EMBED` может также использоваться с `#FOR`, `#LOOP` и/или рекурсивной `#GROUP` для текущего экземпляра символа (если он был зафиксирован). Атрибут `MAP` позволяет Вам заменить описание символа в дереве доступных точек вставки.

Пример:

```
#PROCEDURE(SampleProc,'This is a sample procedure'),WINDOW  
#!Template and target language data declarations for the procedure go here  
#EMBED(%DataSection,'Data Section Source Code Window'),DATA
```

```

                                #!Source code starting in column 1
CODE      !Begin executable code
#EMBED(%SetupProc,'Code Section Source Code Window 1')
                                #!Source code starting in column 3
    here  #!Template and target language executable code for the procedure goes
OPEN(Screen)                                !Open window
ACCEPT                                !Event handler
    CASE SELECTED()                                !Handle field-selection events
    #FOR (%Control)
    OF %Control
        #EMBED(%ScreenFieldSetupEmbed,'Field Selected Embed'),%Control
    #ENDFOR
    END
    CASE ACCEPTED()                                !Handle field-action events
    #FOR (%Control)
    OF %Control
        #EMBED(%ScreenFieldEditEmbed,'Field Accepted Embed'),%Control
    #ENDFOR
    . .
#EMBED(%CustomRoutines,'Code Section Source Code Window 2')
                                #!Source code starting in column 1

```

#AT (вставка текста в точку вставки)

```

#AT ( location [, instances ] ) [, WHERE ( expression ) ]
    statements
#ENDAT

```

#AT	Указывает <i>location</i> для размещения <i>statements</i> .
<i>location</i>	Идентификатор #EMBED. Это может быть #EMBED в процедуре из другого набора шаблонов.
<i>instances</i>	Параметры <i>location</i> для идентификации точки вставки кода для многозначных идентификаторов #EMBED. Можно указывать столько параметров <i>instance</i> , сколько нужно для точной идентификации точки вставки.
WHERE	Определяет, что #AT <i>location</i> указывает только на те точки вставки, где <i>expression</i> истинно.
<i>expression</i>	Выражение, которое определяет размещение.
<i>statements</i>	Операторы языка шаблонов и/или целевого языка.
#ENDAT	Заканчивает структуру.

Структура `#AT` определяет место для генерации операторов. `#AT` может находиться только в секциях шаблонов `#CONTROL`, `#CODE` или `#EXTENSION` и используется в них для того, чтобы генерировать операторы в нескольких точках вставки. Структура `#AT` должна заканчиваться оператором `#ENDAT`.

Фраза `WHERE` позволяет записать выражение, которое укажет на единственный описываемый экземпляр `#EMBED` с атрибутом `symbol`.

Пример:

```
#CONTROL(BrowseList,'Add Browse List controls')
  #AT(%ControlEvent,'?Insert','Accepted')
    #IF (%InsertAllowed)
      Action = AddRecord
    %UpdateProc
    #ENDIF
  #ENDAT
#!
```

См. также: `#EMBED`, `#RESTRICT`, `#CODE`, `#CONTROL`, `#EXTENSION`

#ATSTART (инициализация шаблона)

```
#ATSTART
  statements
#ENDAT
```

#ATSTART	Определяет текст на языке шаблонов, который выполняется перед генерацией <code>#PROCEDURE</code> , <code>#CODE</code> , <code>#CONTROL</code> или <code>#EXTENSION</code> .
<i>statements</i>	Текст на языке шаблонов.
#ENDAT	Заканчивает структуру.

Структура `#ATSTART` определяет текст на языке шаблонов, который должен быть выполнен перед генерацией кода шаблонами `#PROCEDURE`, `#CODE`, `#CONTROL` или `#EXTENSION`. Следовательно, возможны только операторы языка шаблонов. Обычно `#ATSTART` используется для инициализации внутренних переменных шаблона.

Пример:

```
#CONTROL(BrowseList,'Add Browse List controls')
  #ATSTART
    #DECLARE(%ListQueue)
  #ENDAT
```

См. также: `#PROCEDURE`, `#CODE`, `#CONTROL`, `#EXTENSION`

#ATEND (завершение шаблона)

```
#ATEND
  statements
#ENDAT
```

#ATEND Определяет текст на языке шаблонов, который выполняется после генерации #PROCEDURE, #CODE, #CONTROL или #EXTENSION.

statements Текст на языке шаблонов.

#ENDAT Заканчивает структуру.

Структура #ATEND определяет текст на языке шаблонов, который должен быть выполнен после генерации кода шаблонами #PROCEDURE, #CODE, #CONTROL или #EXTENSION. Следовательно, возможны только операторы языка шаблонов. Обычно #ATEND используется для очистки внутренних переменных шаблона.

Пример:

```
#CONTROL (BrowseList,'Add Browse List controls')
  #ATEND
    #SET (%ListQueue,%NULL)
  #ENDAT
```

См. также: #PROCEDURE, #CODE, #CONTROL, #EXTENSION

#EMPTYEMBED (генерация комментария в пустой точке вставки)

```
#EMPTYEMBED( text [, condition ] )
```

#EMPTYEMBED Генерирует комментарий в пустой точке вставки.

text Строковая константа или константное выражение, содержащее текст для размещения в точке вставки.

condition Выражение, при истинности которого разрешается генерация комментария.

Оператор #EMPTYEMBED указывает, что во всех точках вставки, куда пользователь не поместил свой текст, должен быть сгенерирован комментарий. При этом, в тех точках, где либо пользователь разместил текст вставки, либо текст сгенерирован шаблоном, комментарий не появится.

Оператор может находиться только в секциях #PROGRAM или #MODULE. Обычно condition обычно задается с помощью глобального приглашения к вводу (#PROMPT).

В комментариях text для идентификации точки вставки могут использоваться

встроенные символы %EmbedID, %EmbedDescription и %EmbedParameters:

%EmbedID

Символ идентификации текущий точки вставки.

%EmbedDescription

Описание текущей точки вставки

%EmbedParameters

Экземпляр текущей точки вставки в форме списка, разделенного запятыми.

Пример:

```
#EXTENSION(EmptyEmbeds,'Empty Embed Comments'),APPLICATION
```

```
#PROMPT('Generate Empty EMBED Comments',CHECK),%EmptyEmbeds
```

```
#EMPTYEMBED('!Embed: ' & %EmbedDescription & ' ' &
%EmbedParameters,%EmptyEmbeds)
```

См. также: #PREEMBED, #POSTEMBED

#POSTEMBED (генерация комментария в конце точки вставки)

#POSTEMBED(*text* [, *condition*])

#POSTEMBED	Генерирует комментарий после текста, размещенного в точке вставки.
text	Строковая константа или константное выражение, содержащее текст для размещения в точке вставки.
condition	Выражение, при истинности которого разрешается генерация комментария.

Оператор #POSTEMBED указывает, что в конце всех содержащих текст точек вставки должен быть сгенерирован комментарий. Оператор может находиться только в секциях #PROGRAM или #MODULE. Обычно condition обычно задается с помощью глобального приглашения к вводу (#PROMPT).

В комментариях text для идентификации точки вставки могут использоваться встроенные символы %EmbedID, %EmbedDescription и %EmbedParameters:

%EmbedID

Символ идентификации текущий точки вставки.

%EmbedDescription

Описание текущей точки вставки

%EmbedParameters

Экземпляр текущей точки вставки в форме списка, разделенного запятыми.

Пример:

```
#POSTEMBED('! After Embed Point: ' & %EmbedID & ' ' &
EmbedDescription ' ' & %EmbedParameters,%GenerateEmbedComments)
```

См. также: #PREEMBDED, #EMPTYEMBDED

#PREEMBDED (генерация комментария в конце точки вставки)

#PREEMBDED(*text* [, *condition*])

#PREEMBDED	Генерирует комментарий перед текстом, размещенным в точке вставки.
<i>text</i>	Строковая константа или константное выражение, содержащее текст для размещения в точке вставки.
<i>condition</i>	Выражение, при истинности которого разрешается генерация комментария.

Оператор #PREEMBDED указывает, что в начале всех содержащих текст точек вставки должен быть сгенерирован комментарий. Оператор может находиться только в секциях #PROGRAM или #MODULE. Обычно *condition* обычно задается с помощью глобального приглашения к вводу (#PROMPT).

В комментариях *text* для идентификации точки вставки могут использоваться встроенные символы %EmbedID, %EmbedDescription и %EmbedParameters:

%EmbedID

Символ идентификации текущей точки вставки.

%EmbedDescription

Описание текущей точки вставки

%EmbedParameters

Экземпляр текущей точки вставки в форме списка, разделенного запятыми.

Пример:

```
#PREEMBDED('! Before Embed Point: ' & %EmbedID & ' ' & |
EmbedDescription ' ' & %EmbedParameters,%GenerateEmbedComments)
```

См. также: #POSTEMBDED, #EMPTYEMBDED

Поддержка целостности шаблона

#WHERE (указание доступности #CODE в точке вставки)

#WHERE (*embeds*)

#WHERE	Ограничивает обращение к #CODE только теми точками вставки, где допускается генерация такого кода.
<i>embeds</i>	Разделенный запятыми список идентификаторов #EMBDED,

который определяет те точки, где можно применять #CODE для генерации текста.

Оператор #WHERE ограничивает обращение к #CODE только теми точками вставки, где допускается генерация такого текста. В одном #CODE может быть несколько операторов #WHERE, для исчерпывающего задания всех точек вставки, в которых он может применяться. В секции #CODE оцениваются все операторы #WHERE для того, чтобы определить в каких именно точках вставки разрешается использовать #CODE.

Список точек вставки должен содержать разделенные запятыми отдельные идентификаторы #EMBED и диапазоны идентификаторов в форме FirstIdentifier..LastIdentifier, также разделенные запятыми. Список точек вставки может содержать оба типа в произвольной смеси, с тем чтобы определить все нужные точки вставки.

Пример:

```
#CODE (ChangeProperty,'Change control property')
#WHERE(%AfterWindowOpening..%CustomRoutines)
      #!Appropriate everywhere after window open
#PROMPT('Control to change',WINDOWCONTROL),%Field,REQ
#PROMPT('Property to change',@S20),%Property,REQ
#PROMPT('New Value',@S20),%Value,REQ
%Field{%Property} = '%Value'
```

См. также: #EMBED, #RESTRICT, #CODE

#RESTRICT (ограничения на использование секции)

```
#RESTRICT [, WHERE ( expression ) ]
      statements
#ENDRESTRICT
```

#RESTRICT	Устанавливает условия при которых могут использоваться шаблонные секции (#CODE, #CONTROL, #EXTENSION, #PROCEDURE, #PROGRAM или #MODULE).
WHERE	Операторы #RESTRICT (statements) выполняются только тогда, когда expression истинно.
<i>expression</i>	Логическое выражение, ограничивающее выполнение statements структуры #RESTRICT.
<i>statements</i>	Текст на языке шаблонов, приводящий к #ACCEPT или #REJECT использования секции, содержащей структуру #RESTRICT.
#ENDRESTRICT	Заканчивает структуру #RESTRICT.

Структура #RESTRICT обеспечивает механизм ограничения обращения к секции

шаблона (#CODE, #CONTROL, #EXTENSION, #PROCEDURE, #PROGRAM или #MODULE) только теми точками, в которых допускается генерация такого текста. Любая фраза WHERE в секции шаблона оценивается до #RESTRICT.

Оператор #ACCEPT может быть использован для явного указания, что секция может использоваться. Если выполнение оператора #RESTRICT не выходит на оператор #REJECT, то выполняется неявный #ACCEPT. Для исключения секции из использования должен применяться оператор #REJECT. Как #ACCEPT, так и #REJECT операторы немедленно завершают обработку структуры #RESTRICT.

Пример:

```
#CODE (ChangeControlSize,'Change control size')
#RESTRICT
  #CASE (%ControlType)
  #OF ('LIST')
    #REJECT
  #OF ('BUTTON')
    #REJECT
  #ELSE
    #ACCEPT
  #ENDCASE
#ENDRESTRICT
#PROMPT ('New Width',@n04),%NewWidth
#PROMPT ('New Height',@n04),%NewHeight
%Control{PROP:Width} = %NewWidth
%Control{PROP:Height} = %NewHeight
См. также:#ACCEPT, #REJECT
```

#ACCEPT (разрешение на использование секции)

#ACCEPT

Оператор #ACCEPT завершает обработку структуры #RESTRICT, указывая на доступность секции шаблона (#CODE, #CONTROL, #EXTENSION, #PROCEDURE, #PROGRAM или #MODULE).

Структура #RESTRICT содержит операторы языка шаблонов, которые оценивают свойства генерируемого секцией текста. Оператор #ACCEPT используется для явного объявления что генерируется допустимый текст. Если выполнение оператора #RESTRICT не выходит на оператор #REJECT, то выполняется неявный #ACCEPT. Для исключения секции из использования должен применяться оператор #REJECT. Как #ACCEPT, так и #REJECT операторы немедленно завершают обработку структуры #RESTRICT.

Пример:


```
#CODE (ChangeControlSize,'Change control size')
#WHERE (%EventHandling)
#RESTRICT
#CASE (%ControlType)
#OF 'LIST'
#REJECT
#OF 'BUTTON'
#REJECT
#ELSE
#ACCEPT
#ENDCASE
#ENDRESTRICT
#PROMPT ('New Width',@n04),%NewWidth
#PROMPT ('New Height',@n04),%NewHeight
%Control{PROP:Width} = %NewWidth
%Control{PROP:Height} = %NewHeight
См. также: #RESTRICT, #REJECT
```

#REJECT (запрет на использование секции)

#REJECT

Оператор **#REJECT** завершает обработку структуры **#RESTRICT**, указывая на недоступность секции шаблона (**#CODE**, **#CONTROL**, **#EXTENSION**, **#PROCEDURE**, **#PROGRAM** или **#MODULE**).

Структура **#RESTRICT** содержит операторы языка шаблонов, которые оценивают свойства генерируемого секцией текста. Оператор **#ACCEPT** используется для явного объявления, что генерируется допустимый текст. Если выполнение оператора **#RESTRICT** не выходит на оператор **#REJECT**, то выполняется неявный **#ACCEPT**. Для исключения секции из использования должен применяться оператор **#REJECT**. Как **#ACCEPT** так и **#REJECT** операторы немедленно завершают обработку структуры **#RESTRICT**.

Пример:

```
#CODE (ChangeControlSize,'Change control size')
#WHERE (%EventHandling)
#RESTRICT
#CASE (%ControlType)
#OF 'LIST'
#REJECT
#OF 'BUTTON'
#REJECT
```

```
#ELSE
    #ACCEPT
#ENDCASE
#ENDRESTRICT
#PROMPT ('New Width',@n04),%NewWidth
#PROMPT ('New Height',@n04),%NewHeight
%Control{PROP:Width} = %NewWidth
%Control{PROP:Height} = %NewHeight
См. также:#RESTRICT, #ACCEPT
```

Стандартные структуры и шаблонные переменные

Стандартные данные и тексты

#WINDOWS (стандартная оконная структура)
#REPORTS (стандартная структура документа)
#LOCALDATA (стандартные объявления локальных данных)
#GLOBALDATA (стандартные объявления глобальных данных)
#DEFAULT (стандартная начальная процедура)

Операторы управления символами

#DECLARE (объявление определяемого символа)
#ALIAS (обращение к символу из другого экземпляра)
#ADD (добавление к многозначному символу)
#DELETE (удаление экземпляра многозначного символа)
#DELETEALL (удаление экземпляров многозначного символа)
#PURGE (удаление экземпляров однозначного или многозначного символа)
#CLEAR (очистка однозначного символа)
#FREE (очистка многозначного символа)
#FIX (фиксация многозначного символа)
#FIND (“суперфиксация” многозначных символов)
#SELECT (фиксация многозначного символа)
#SET (присваивание значения определяемому символу)
#UNFIX (расфиксация многозначного символа)
Атрибуты оператора #DECLARE
UNIQUE (запрет повторений)
SAVE (сохранение значений символа между генерациями)

Стандартные данные и тексты

#WINDOWS (стандартная оконная структура)

```
#WINDOWS
    structures
#ENDWINDOWS
```

#WINDOWS Начинает секцию структуры стандартного окна.
structures Стандартные структуры APPLICATION или WINDOW.
#ENDWINDOWS Заканчивает секцию стандартного окна.

Структура #WINDOWS содержит стандартные структуры APPLICATION или WINDOW процедурного шаблона. Стандартные оконные структуры определяют начальное состояние разрабатываемого в процедуре окна.

Структура #WINDOWS может содержать несколько structures, одна из которых может быть выбрана в качестве начального состояния разрабатываемого окна. Если имеется более одной структуры, Генератор Приложений показывает список доступных структур при первом редактировании окна процедуры. Имена окон, которые появляются в списке Генератора приложений берутся из предшествующей строки комментария, начинающейся с двух восклицательных знаков и правой угловой скобки (!!<).

Если процедурный шаблон содержит процедуру #DEFAULT, структура #WINDOWS становится не нужна, так как стандартное окно уже входит в #DEFAULT. Поэтому при первом редактировании окна список не появляется.

Пример:

```
#WINDOWS
!!> Window
Label WINDOW ('Caption'),AT(0,0,100,100)
    END
!!> Window with OK & Cancel
Label WINDOW ('Caption'),AT(0,1,185,92)
    BUTTON('OK'),AT(144,10,35,14),DEFAULT,USE(?Ok)
    BUTTON('Cancel'),AT(144,28,36,14),USE(?Cancel)
    END
#ENDWINDOWS
```

#REPORTS (стандартная структура документа)**#REPORTS***structures***#ENDREPORTS**

#REPORTS Начинает секцию структуры стандартного документа.
structures Стандартные структуры REPORT.
#ENDREPORTS Заканчивает секцию стандартного документа.

Структура **#REPORTS** содержит стандартные структуры **REPORT** процедурного шаблона. Стандартные структуры документов определяют начальное состояние разрабатываемого в процедуре документа.

Структура **#REPORTS** может содержать несколько *structures*, одна из которых может быть выбрана в качестве начального состояния разрабатываемого документа. Если имеется более одной структуры, Генератор Приложений показывает список доступных структур при первом редактировании документа процедуры. Имена документов, которые появляются в списке Генератора приложений, берутся из предшествующей строки комментария, начинающейся с двух восклицательных знаков и правой угловой скобки (!!<).

Если процедурный шаблон содержит процедуру **#DEFAULT**, структура **#REPORT** становится не нужна, так как стандартный документ уже входит в **#DEFAULT**. Поэтому при первом редактировании документа список не появляется.

:

#REPORTS

!!> Report

Label REPORT,AT(1000,2500,6000,6000),THOUS

HEADER,AT(1000,1000,6000,1000)

END

Detail DETAIL

END

FOOTER,AT(1000,10000,6000,1000)

END

FORM,AT(1000,1000,6000,9000)

END

END

#ENDREPORTS

#LOCALDATA (стандартные объявления локальных данных)

```
#LOCALDATA
    declarations
#ENDLOCALDATA
```

#LOCALDATA Начинает секцию стандартных локальных данных.
declarations Объявления стандартных данных.
#ENDLOCALDATAЗаканчивает секцию стандартных локальных данных.

Структура **#LOCALDATA** содержит стандартные объявления локальных данных процедуры, генерируемой процедурным шаблоном. Структура **#LOCALDATA** может находиться в шаблонных секциях **#PROCEDURE**, **#CODE**, **#CONTROL** или **#EXTENSION**. Declarations размещаются в генерируемой процедуре между ключевыми словами **PROCEDURE** (или **FUNCTION**) и **CODE**.

Пример:

```
#LOCALDATA
Action      B Y T E           !Disk action variable
TempFileCTRING(65)         !Temporary filename variable
#ENDLOCALDATA
```

#GLOBALDATA (стандартные объявления глобальных данных)

```
#GLOBALDATA
    declarations
#ENDGLOBALDATA
```

#GLOBALDATA Начинает секцию стандартных глобальных данных.
declarations Объявления стандартных данных.
#ENDGLOBALDATA Заканчивает секцию стандартных глобальных данных.

Структура **#GLOBALDATA** содержит стандартные объявления глобальных данных процедуры, генерируемой процедурным шаблоном. Структура **#GLOBALDATA** может находиться в шаблонных секциях **#PROCEDURE**, **#CODE**, **#CONTROL** или **#EXTENSION**. declarations размещаются в секции глобальных данных генерируемого исходного текста.

Пример:

```
#GLOBALDATA
Action      B Y T E           !Disk action variable
TempFileCTRING(65)         !Temporary filename variable
```

#ENDGLOBALDATA

#DEFAULT (стандартная начальная процедура)

#DEFAULT

procedure

#ENDDEFAULT

#DEFAULT	Начинает секцию стандартной начальной процедуры.
procedure	Стандартная процедура в формате .TXA.
#ENDDEFAULT	Заканчивает секцию стандартной начальной процедуры.

Структура #DEFAULT содержит единственное объявление стандартной процедуры в формате .TXA, как генерирует функция Export Генератора Приложений. Структура #DEFAULT может находиться только в конце секции #PROCEDURE шаблона. В одной процедуре можно иметь несколько структур DEFAULT. Включаемая .TXA секция должна содержать процедуру того же типа, что и предыдущий оператор #PROCEDURE. Рекомендуется создавать структуры #DEFAULT, редактируя стандартные процедуры в регистре шаблонов и экспортируя шаблон в форме текста.

Пример:

```
#DEFAULT
NAME DefaultForm
[COMMON]
DESCRIPTION 'Default record update'
FROM Clarion Form
[PROMPTS]
%WindowOperationMode STRING ('Use WINDOW setting')
%INISaveWindow LONG (1)
[ADDITION]
NAME Clarion SaveButton
[INSTANCE]
INSTANCE 1
PROCPROP
[PROMPTS]
%InsertAllowed LONG (1)
%InsertMessage @S30 ('Record will be Added')
%ChangeAllowed LONG (1)
%ChangeMessage @S30 ('Record will be Changed')
%DeleteAllowed LONG (1)
%DeleteMessage @S30 ('Record will be Deleted')
%MessageHeader LONG (0)
[ADDITION]
```

```

NAME Clarion CancelButton
[INSTANCE]
INSTANCE 2
[WINDOW]
FormWindow WINDOW
('Update records...'),AT(18,5,289,159),CENTER,SYSTEM,GRAY,MDI

BUTTON('OK'),AT(5,140,40,12),USE(?OK),#SEQ(1),#ORIG(?OK),#LINK(?Cancel)

BUTTON('Cancel'),AT(50,140,40,12),USE(?Cancel),#SEQ(2),#ORIG(?Cancel)
    STRING (@s40),AT(95,140,,),USE(ActionMessage)
    END
#ENDDEFAULT

```

Операторы управления символами

#DECLARE (объявление определяемого символа)

```
#DECLARE ( symbol [, parentsymbol [,type] ] ) [, MULTI ] [, UNIQUE ] [, SAVE]
```

#DECLARE	Явно объявляет определяемый пользователем символ.
<i>symbol</i>	Идентификатор определяемого символа. Он должен отвечать всем требованиям к определяемым символам. Он не может совпадать с символом в #PROMPT или переменной в той же самой области видимости.
<i>parentsymbol</i>	Указывает на предка этого символа, определяя, что его значение зависит от текущего значения другого символа. Это должен быть многозначный символ. Можно указать более одного предка символа, если символ зависит от набора символов. Это позволяет неявно объявлять многомерные массивы.
<i>type</i>	Тип данных parentsymbol: LONG, REAL или STRING. Если опущено, то используется STRING.
MULTI	Указывает, что <i>symbol</i> может содержать множество значений.
UNIQUE	Указывает, что многозначный символ не может содержать повторяющиеся значения. Значения хранятся в порядке возрастания. Это неявно объявляет символ как многозначный, и поэтому атрибут MULTI не требуется.
SAVE	Указывает, что значение(я) символа сохраняется от генерации к генерации. Символ с атрибутом SAVE может быть объявлен только в секции #APPLICATION.

Оператор #DECLARE явно объявляет определяемый пользователем символ. Он может содержать единственное значение или множество значений. Все определяемые символы должны быть явно объявлены в #DECLARE, за исключением тех, которые объявляются

в операторах #PROMPT или являются параметрами #GROUP.

Атрибут MULTI объявляет символ как многозначный. Это позволяет применять к символу операторы #FIX, #FOR, #ADD, #DELETE, #SELECT и #FREE.

Многозначный определяемый символ может рассматриваться как массив или очередь. Как к элементу массива, к отдельному экземпляру многозначного определяемого символа в выражении можно адресоваться с помощью %symbol[1].

Атрибут UNIQUE гарантирует, что все экземпляры многозначного символа будут уникальны и отсортированы в порядке возрастания. Когда указано UNIQUE, MULTI не требуется. Оператор #ADD выстраивает значения символа в порядке возрастания и разрешает при добавлении иметь только единственный экземпляр каждого значения.

Если оператор #DECLARE содержит один или несколько параметров parentsymbol, определяемый символ зависит от parentsymbol-ов. Это означает, что для каждого экземпляра parentsymbol имеется отдельный экземпляр (или экземпляры, если многозначный) symbol. Если нет параметров parentsymbol, то symbol независим.

#DECLARE может применяться для создания зависимых символов. Parentsymbol должен быть многозначным символом, независимо от того, является он встроенным или определяемым.

Атрибут SAVE вызывает сохранение значения(ий) символа(ов) в конце генерации исходного текста и их загрузки при выполнении оператора #DECLARE в начале следующего цикла генерации текста. Символ с атрибутом SAVE может быть объявлен только в секции #APPLICATION.

Пример:

```
#APPLICATION ('Sample One')
#DECLARE (%UserSymbol),SAVE                #!Value saved after generation
                                           #! and restored for next generation
#DECLARE (%ModuleFile,%Module),UNIQUE,MULTI  #!Level-1 dependent symbol
bol
#DECLARE (%ModuleFilePut,%ModuleFile)        #!Level-2 dependent symbol
#DECLARE (%ModuleFileDelete,%ModuleFile)     #!Second Level-2 dependent symbol
bol
```

См. также: #FIX, #FOR, #ADD, #DELETE, #FREE

#ALIAS (обращение к символу из другого экземпляра)

#ALIAS (*oldsymbol* , *newsymbol* [, *instance*])

#ALIAS	Переопределяет определяемый пользователем символ.
<i>oldsymbol</i>	Идентификатор переопределяемого символа. Он должен отвечать всем требованиям к определяемым символам. Он не может совпадать с символом в #PROMPT или переменной в той же самой области видимости.
<i>newsymbol</i>	Объявляет новое имя для <i>oldsymbol</i> .
<i>instance</i>	Выражение, определяющее экземпляр добавления, содержащего <i>oldsymbol</i> .

Оператор #ALIAS переопределяет определяемый пользователем символ, объявленный в шаблонном диалоговом элементе #CODE, #CONTROL или #EXTENSION для использования в другом шаблоне.

Пример:

```
#EXTENSION (GlobalSecurity,'Global Password Check'),APPLICATION
#DECLARE (%PasswordFile)
#DECLARE (%PasswordFileKey)
```

```
#EXTENSION (LocalSecurity,'Local Procedure Password Check'),PROCEDURE
#ALIAS (%PasswordFile,%PswdFile,'GlobalSecurity(Clarion)')
#ALIAS (%PasswordFileKey,%PswdFileKey,'GlobalSecurity(Clarion)')
```

См. также: #CODE, #CONTROL, #EXTENSION

#ADD (добавление к многозначному символу)

#ADD (*symbol* , *expression* [, *position*])

#ADD	Добавляет экземпляр к определяемому пользователем символу.
<i>symbol</i>	Многозначный определяемый пользователем символ.
<i>expression</i>	Выражение, определяющее значение, которое должно быть помещено в новый экземпляр символа.
<i>position</i>	Числовая константа или символ, содержащий номер экземпляра для добавления символа. Экземпляры нумеруются начиная с единицы (1). Если <i>position</i> больше, чем количество уже существующих экземпляров плюс один, новый экземпляр добавляется в конец, и не создается никаких внедряющихся экземпляров.

Добавляет значение к многозначному определяемому пользователем символу. Выполняется подразумеваемый #FIX на этот экземпляр символа. Если символ не

является многозначным определяемым пользователем символом, фиксируется ошибка генерации текста.

Если символ был объявлен с атрибутом UNIQUE, то #ADD является операцией объединения с существующим набором значений символа. Может существовать только один экземпляр добавляемого значения. Атрибут UNIQUE также определяет, что #ADD это упорядоченная вставка в существующий набор значений символа. После каждого #ADD все значения символа будут отсортированы.

Если символ был объявлен без атрибута UNIQUE, то разрешаются повторяющиеся значения. Новое значение добавляется в конец списка и может повторяться. Если символ повторяется, то наследуются все зависимые экземпляры потомка.

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI,UNIQUE  #!Declare unique multi-val-
ued symbol
#FIX (%File,%Primary)                      #!Build list of all file prefixes in proc
#ADD (%ProcFilesPrefix,%FilePre)          #!Start with primary file
#FOR (%Secondary)                          #!Then add all secondary files
  #FIX (%File,%Secondary)
  #ADD (%ProcFilesPrefix,%FilePre)
#ENDFOR
См. также: #DECLARE
```

#DELETE (удаление экземпляра многозначного символа)

#DELETE (*symbol* [, *position*])

#DELETE	Уничтожает значение в одном экземпляре многозначного определяемого пользователем символа.
<i>symbol</i>	Многозначный определяемый пользователем символ.
<i>position</i>	Числовая константа или символ, содержащий номер экземпляра символа. Экземпляры нумеруются начиная с единицы (1). Если опущено, удаляется текущий зафиксированный экземпляр.

Оператор #DELETE стирает значение в одном экземпляре многозначного определяемого пользователем символа. Если есть какие либо символы, зависящие от удаляемого символа, они тоже очищаются. Если это последний экземпляр символа, экземпляр удаляется. Текущий номер экземпляра на котором зафиксирован символ можно получить, используя встроенную шаблонную функцию INSTANCE(%symbol).

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI      #!Declare multi-valued symbol
```

```
#ADD (%ProcFilesPrefix,'SAV')      #!Add a value
#ADD (%ProcFilesPrefix,'BAK')      #!Add a value
#ADD (%ProcFilesPrefix,'PRE')      #!Add a value
#ADD (%ProcFilesPrefix,'QUE')      #!Add a value
#!%ProcFilesPrefix contains: SAV, BAK, PRE, QUE
#DELETE (%ProcFilesPrefix,1)        #!Delete first value (SAV)
#!%ProcFilesPrefix contains: BAK, PRE, QUE
#FIX (%ProcFilesPrefix,'PRE')       #!Fix to a value
#DELETE (%ProcFilesPrefix)          #!Delete it
#!%ProcFilesPrefix contains: BAK, QUE
См. также: #DECLARE, #ADD
```

#DELETEALL (удаление экземпляров многозначного символа)

#DELETEALL(*symbol*, *expression*)

expression Выражение, которое задает удаляемые экземпляры.

Оператор #DELETEALL удаляет все значения, которые удовлетворяют выражению *expression*.

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI  #!Declare multi-valued symbol
#ADD (%ProcFilesPrefix,'SAV')      #!Add a value
#ADD (%ProcFilesPrefix,'BAK')      #!Add a value
#ADD (%ProcFilesPrefix,'PRE')      #!Add a value
#ADD (%ProcFilesPrefix,'BAK')      #!Add a value
#ADD (%ProcFilesPrefix,'QUE')      #!Add a value
#!%ProcFilesPrefix contains: SAV, BAK, PRE, BAK, QUE
#DELETEALL(%ProcFilesPrefix,'BAK') #!Delete all BAK instances
#!%ProcFilesPrefix now contains: SAV, PRE, QUE
См. также: #DECLARE, #ADD
```

#PURGE (удаление экземпляров однозначного или многозначного символа)

#PURGE(*symbol*)

#PURGE Уничтожает все значения во всех экземплярах определяемого пользователем символа.

symbol Определяемый пользователем символ.

Оператор #PURGE уничтожает все значения *symbol*. Если есть какие-либо

зависимые от этого символа другие символы, они тоже очищаются. Если symbol зависит от многозначного символа, все экземпляры этого зависимого символа очищаются для всех экземпляров многозначного символа, от которого он зависит.

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI      #!Declare multi-valued symbol
#ADD (%ProcFilesPrefix,'SAV')           #!Add a value
#ADD (%ProcFilesPrefix,'BAK')           #!Add a value
#ADD (%ProcFilesPrefix,'PRE')           #!Add a value
#ADD (%ProcFilesPrefix,'BAK')           #!Add a value
#ADD (%ProcFilesPrefix,'QUE')           #!Add a value
#!%ProcFilesPrefix contains: SAV, BAK, PRE, BAK, QUE
#PURGE(%ProcFilesPrefix)                 #!Delete all instances
См. также: #DECLARE, #ADD
```

#CLEAR (очистка однозначного символа)

#CLEAR(*symbol*)

#CLEAR Уничтожает значение однозначного определяемого пользователем символа.

symbol Однозначный определяемый пользователем символ.

Оператор #CLEAR уничтожает значение в однозначном определяемом пользователем символе. Этот оператор делает приблизительно тоже самое, что и применение #SET для присваивания пустого значения символу, за исключением того, что он эффективнее.

Пример:

```
#DECLARE (%SomeSymbol)                  #!Declare symbol
#SET(%SomeSymbol,'Value')               #!Assign a value
#!%SomeSymbol now contains: 'Value'
#CLEAR(%SomeSymbol)                     #!Clear value
#!%SomeSymbol now contains: ''
См. также: #DECLARE, #ADD
```

#FREE (очистка многозначного символа)

#FREE(*symbol*)

#FREE Уничтожает все экземпляры многозначного определяемого пользователем символа.

symbol Многозначный определяемый пользователем символ.

Оператор #FREE уничтожает все экземпляры многозначного определяемого пользователем символа. Если есть какие-либо символы, зависящие от symbol, они тоже очищаются.

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI      #!Declare multi-valued symbol
#ADD (%ProcFilesPrefix,'SAV')           #!Add a value
#ADD (%ProcFilesPrefix,'BAK')           #!Add a value
#ADD (%ProcFilesPrefix,'PRE')           #!Add a value
#ADD (%ProcFilesPrefix,'BAK')           #!Add a value
#ADD (%ProcFilesPrefix,'QUE')           #!Add a value
#!%ProcFilesPrefix contains: SAV, BAK, PRE, BAK, QUE
#DELETEALL(%ProcFilesPrefix,'BAK')      #!Delete all BAK instances
#!%ProcFilesPrefix now contains: SAV, PRE, QUE
#FREE(%ProcFilesPrefix)                  #!Free the symbol
#!%ProcFilesPrefix now contains nothing
См. также: #DECLARE, #ADD
```

#FIX (фиксация многозначного символа)

#FIX (*symbol*, *fixsymbol*)

#FIX	Фиксирует многозначный символ на значении его отдельного экземпляра.
<i>symbol</i>	Многозначный символ.
<i>fixsymbol</i>	Символ или выражение, содержащее фиксируемое значение.

Оператор #FIX фиксирует текущее значение многозначного символа на значении, содержащемся в fixsymbol. Это делается для того, чтобы можно было ссылаться на один их экземпляров символа вне структуры #FOR или чтобы можно было сослаться на зависящие от многозначного символа другие символы.

В fixsymbol должен содержаться правильный экземпляр одного из множества значений symbol. Если fixsymbol не содержит надлежащего экземпляра, symbol очищается и при ссылке на него не содержит никакого значения. До тех пор, пока для добавления нового значения и его фиксации не применялся #ADD, для того, чтобы установить значение символа, и он содержал какое-либо значение, используемое при обработке шаблона вне цикла #FOR, должны применяться #FIX или #SELECT.

#FIX полностью не зависит от #FOR, в том смысле, что #FOR всегда перебирает все экземпляры символа, независимо от того выполнялся ли для этого символа предшествующий #FIX или нет. Если перед циклом #FOR для этого же символа выполнен #FIX, символ будет ссылаться на то же самое fixvalue по окончании #FOR.

Если #FIX используется внутри структуры #FOR, область действия #FIX ограничивается структурой #FOR в которой он используется. Это не изменяет итерационного значения в #FOR, если и #FOR и #FIX применяются к одному и тому же символу.

Пример:

```
#SET(%OneFile,'HEADER')      #! Put values into two User-defined sym-
bols                          bols
#SET(%TwoFile,'DETAIL')
#FIX (%File,%OneFile)        #! %File refers to 'HEADER'
#FOR (%File)                  #! %File iteratively refers to all file names
    #FIX (%File,%TwoFile)    #! %File refers to 'DETAIL'
#ENDFOR
#! %File refers to 'HEADER' again
См. также: #SELECT
```

#FIND (“суперфиксация” многозначных символов)

#FIND (*symbol*, *fixsymbol* [, *limit*])

#FIND	Фиксирует все многозначные родительские символы на значениях, которые указывают на единственный дочерний экземпляр.
<i>symbol</i>	Многозначный символ.
<i>fixsymbol</i>	Символ или выражение, содержащее фиксируемое значение.
<i>limit</i>	Родительский символ, ограничивающий область поиска потомками символа <i>limit</i> .

Оператор #FIND находит первый экземпляр *fixsymbol*, содержащийся в *symbol*, затем фиксирует его и все “родительские” символы от которых зависит *symbol* на значениях, которые “указывают на” значение *fixsymbol*, содержащееся в *symbol*.

Например, предположим что %ControlUse содержит CUS:Name. Оператор #FIND(%Field,%ControlUse,%Control) :

- Находит первый экземпляр %Field, который соответствует текущему значению %ControlUse (первый экземпляр CUS:Name в %Field) в текущей процедуре.
- Фиксирует в %Field это значение (CUS:Name).
- Фиксирует в %File имя файла, содержащего это поле (Customer).
- Это позволяет в тексте шаблона ссылаться на другие символы, зависящие от %File (например %FilePre, чтобы получить префикс этого файла).

В *fixsymbol* должен содержаться правильный экземпляр одного из множества значений *symbol*. Если *fixsymbol* не содержит надлежащего экземпляра, *symbol* очищается и при

ссылке на него не содержит никакого значения.

Пример:

```
#FIND (%Field,%ControlUse)#!Fixes %Field and %File to %ControlUse parents
```

См. также: #SELECT, #FIX

#SELECT (фиксация многозначного символа)

#SELECT (*symbol*, *instance*)

#SELECT Фиксирует многозначный символ на экземпляре с указанным номером.

symbol Многозначный символ.

instance Выражение, содержащее номер фиксируемого экземпляра.

Оператор #SELECT фиксирует текущее значение многозначного символа на экземпляре с указанным номером. Результат #SELECT в точности тот же самый, что и у #FIX. Все экземпляры многозначного символа нумеруются начиная с единицы (1).

В *instance* должен содержаться правильный порядковый номер одного из значений многозначного символа. Если *instance* неверно, *symbol* очищается и при ссылке не содержит никаких значений. Встроенная шаблонная функция INSTANCE возвращает номер экземпляра.

До тех пор, пока для добавления нового значения и его фиксации не применялся #ADD, для того, чтобы установить значение символа и он содержал какое-либо значение, используемое при обработке шаблона вне цикла #FOR, должны применяться #FIX или #SELECT.

Пример:

```
#SELECT (%File,1)
```

```
#!Fix to first %File instance
```

#SET (присваивание значения определяемому символу)

#SET (*symbol*, *value*)

#SET Присваивает значение однозначному определяемому пользователем символу.

symbol Однозначный определяемый пользователем символ. Он должен быть предварительно объявлен оператором #DECLARE.

value Встроенный или определяемый пользователем символ, строковая константа или выражение.

Оператор #SET присваивает значение символу. Если параметр value содержит выражение, можно выполнить вычисления во время генерации текста. В выражении могут использоваться любые описанные в Language Reference арифметические и логические операции. Если в выражении используется операция деления по модулю (%) за ней должен следовать по крайней мере один пробел (чтобы она явно отличалась от символов шаблона). Логические выражения всегда сводятся к 1 (Истина) или 0 (Ложь). Разрешается вызов функций языка Clarion (разрешенных в EVALUATE()) и встроенных шаблонных функций.

Пример:

```
#SET (%NetworkApp,'Network')
#SET (%MySymbol,%Primary)
#FOR (%File)
    #SET (%FilesCounter,%FilesCounter + 1)
%FileStructure
#ENDFOR
```

#UNFIX (расфиксация многозначного символа)

#UNFIX (*symbol*)

#UNFIX	Расфиксация многозначного символа.
symbol	Многозначный символ.

Оператор #UNFIX расфиксирует текущее значение многозначного символа. Если расфиксированный символ применяется вне структуры #LOOP, он не имеет значения и не может указывать на другие, зависящие от него символы.

Пример:

#SET (%OneFile,'HEADER')	#! Put values into two User-defined symbols
#SET (%TwoFile,'DETAIL')	
#FIX (%File,%OneFile)	#! %File refers to 'HEADER'
#FOR (%File)	#! %File iteratively refers to all file names
#FIX (%File,%TwoFile)	#! %File refers to 'DETAIL'
#ENDFOR	
#! %File refers to 'HEADER' again	
#UNFIX (%File)	#! %File refers to no specific value

Атрибуты оператора #DECLARE

UNIQUE (запрет повторений)

UNIQUE

Атрибут UNIQUE в операторе #DECLARE указывает, что объявляемый многозначный символ не может содержать повторяющиеся значения. Для соблюдения этого требования оператор #ADD всегда добавляет экземпляры символа в порядке возрастания.

Пример:

```
#DECLARE (%ProcFilesPrefix),MULTI,UNIQUE #!Declare unique multi-val-
ued symbol
#FIX (%File,%Primary)           #!Build list of all file prefixes in proc
#ADD (%ProcFilesPrefix,%FilePre) #!Start with primary file
#FOR (%Secondary)               #!Then add all secondary files
    #FIX (%File,%Secondary)
    #ADD (%ProcFilesPrefix,%FilePre)
#ENDFOR
См. также: #DECLARE
```

SAVE (сохранение значений символа между генерациями)

SAVE

Атрибут SAVE в операторе #DECLARE вызывает сохранение значения(й) объявляемого символа в конце генерации и их восстановление перед началом следующей генерации. Оператор #DECLARE с атрибутом SAVE может находиться только в секции #APPLICATION.

Пример:

```
#APPLICATION ('Sample One')
#DECLARE (%UserSymbol),SAVE      #!Value saved after generation
                                  #! and restored for next generation
#DECLARE (%ModuleFile,%Module),UNIQUE,MULTI #!Level-1 dependent sym-
bol
#DECLARE (%ModuleFilePut,%ModuleFile)      #!Level-2 dependent symbol
#DECLARE (%ModuleFileDelete,%ModuleFile)    #!Second Level-2 dependent sym-
bol
См. также: #DECLARE
```

Ввод программиста

Операторы ввода и проверки

#PROMPT (приглашение к вводу)
#VALIDATE (проверка ввода)
#DISPLAY (вывод)
#ENABLE (включение/выключение элементов диалога)
#BOXED (группа элементов диалога)
#BUTTON (вызов новой страницы диалога)
#FIELD (ввод для диалоговых элементов окна)

Варианты type в #PROMPT

CHECK (переключатель)
COMPONENT (список полей ключа)
CONTROL (список полей окна)
DROP (выпадающий список)
EMBED (ввод текста в точку вставки)
EMBEDBUTTON (ввод текста в точку вставки)
FIELD (список полей данных)
FILE (список файлов)
FORMAT (вызов форматера списковой структуры)
FROM (список значений символа)
KEY (список ключей)
KEYCODE (список кодов клавиш)
OPTION (вывод радиокнопок)
PROCEDURE (добавление процедуры к дереву процедур)
RADIO (единичная радиокнопка)
SPIN (элемент диалога с “прокруткой”)

Операторы вывода и оформления.

#BOXED (группа элементов диалога)
#DISPLAY (вывод)
#IMAGE (вывести графическое изображение)
#SHEET (объявляет группу диалоговых элементов #TAB)
#TAB (объявляет страницу диалогового элемента #SHEET)

Операторы ввода и проверки

#PROMPT (приглашение к вводу)

```
#PROMPT ( string, type ) [, symbol ] [, REQ ] [, DEFAULT( value ) ]
          [, MULTI ( description ) ] [, INLINE ] [, SELECTION( description ) ]
[, CHOICE ]
          [,ICON(file)][,AT()][,PROMPTAT()]
```

#PROMPT

string

Запрашивает ввод у программиста.

Строковая константа, содержащая текст, который выводится в качестве приглашения к вводу. Он может содержать амперсанд (&), указывающий на “горячую” клавишу, которая вместе с клавишей ALT позволяет выбрать поле в окне свойств.

type

Шаблон изображения или ключевое слово.

symbol

Определяемый символ, который получит результат ввода. #PROMPT с type RADIO или EMBED не могут иметь symbol, во всех остальных случаях symbol должен быть.

REQ

Указывает, что ввод не может быть пустым или нулевым.

DEFAULT

Определяет начальное значение (которое может быть изменено).

value

Строковая константа, содержащая начальное значение.

MULTI

Указывает, что программист может ввести несколько значений для #PROMPT. Диалоговый элемент принимает вид кнопки, которая открывает списковую структуру, позволяющую программисту ввести несколько значений.

description

Строковая константа, содержащая надпись на кнопке и в заголовке списка запрашиваемых величин.

INLINE

Указывает, что #PROMPT имеет вид списка с кнопками редактирования, что позволяет пользователю ввести множество значений символа.

При этом должен быть указан и атрибут MULTI.

SELECTION

Указывает, что программист должен выбрать значение для #PROMPT из списка, заданного атрибутом FROM. Приглашение имеет вид кнопки, которая вызывает всплывающий список, из которого программист может выбрать значение, если только не указан и атрибут INLINE.

CHOICE

Указывает, что symbol получает значение не строки, а порядкового номера выбранного RADIO в группе OPTION.

ICON

Определяет пиктограмму кнопки для #PROMPT с атрибутом MULTI.

file

Строковая константа, содержащая спецификацию файла .ICO с пиктограммой кнопки.

AT

Указывает положение в окне области ввода относительно первого

диалогового элемента, размещенного шаблоном на экране (за исключением стандартных диалоговых элементов в окне свойств процедуры). Этот атрибут использует те же самые параметры, что и атрибут AT в языке Clarion.

PROMPTAT

Указывает на положение в окне строки приглашения относительно первого диалогового элемента, размещенного шаблоном на экране (за исключением стандартных диалоговых элементов в окне свойств процедуры). Этот атрибут использует те же самые параметры, что и атрибут AT в языке Clarion.

Оператор #PROMPT запрашивает ввод у программиста. Оператор #PROMPT может находиться в секциях шаблона #APPLICATION, #PROCEDURE, #CODE, #CONTROL, #EXTENSION или #FIELD. Он не может находиться в секциях #PROGRAM, #MODULE, #TEMPLATE или #GROUP.

Когда #PROMPT размещается в секции шаблона, его приглашение и связанное с ним поле ввода размещаются:

Имя секции

#APPLICATION
#PROCEDURE
#CODE
#CONTROL

#EXTENSION
#FIELD

Имя окна

Global Settings (глобальные установки)
Procedure Properties (свойства процедуры)
Embeds Dialog (точки вставки)
Actions Dialog и Controls Dialog (действия и диалоговые элементы)
Extensions Dialog (распределенные шаблоны)
Actions Dialog (действия)

Параметр type может содержать шаблон изображения для форматирования ввода или одно из следующих ключевых слов:

PROCEDURE	Метка процедуры
FILE	Метка файла данных
KEY	Метка ключа (может быть ограничена одним файлом)
COMPONENT	Метка поля, входящего в ключ (может быть ограничена одним ключом)
FIELD	Метка поля в файле данных (может быть ограничена одним файлом)
FORMAT	Вызывает формater спискового диалогового элемента.
PICTURE	Вызывает формater шаблонов ввода.
DROP	Создает выпадающий список указанных в параметре вариантов
KEYCODE	Код клавиши или его EQUATE
OPTION	Группирует радиокнопки
RADIO	Создает радиокнопку

CHECK	Создает переключатель
CONTROL	Список диалоговых элементов окна
FROM	Создает выпадающий список из значений указанного в параметре символа
EMBED	Позволяет пользователю редактировать указанную точку вставки
EMBEDBUTTON	Позволяет пользователю редактировать указанную точку вставки
SPIN	Создает элемент диалога типа SPIN
OPENDIALOG	Вызывает стандартный диалог Windows Open File.
SAVEDIALOG	Вызывает стандартный диалог Windows Save File.

Для всех типов, за исключением RADIO и CHECK (и элементов с атрибутом MULTI), строка приглашения выводится непосредственно слева от области ввода данных.

Оператор #PROMPT с атрибутом REQ не может быть оставлен пустым или с нулевым значением - это обязательное для ввода поле. Атрибут DEFAULT может применяться для того, чтобы указать в #PROMPT начальное значение, которое может быть переопределено во время проектирования.

Оператор #PROMPT с type RADIO создает радиокнопку для непосредственно предшествующего #PROMPT с type OPTION. В одном OPTION может быть много RADIO. При выборе каждая string у RADIO размещается в ближайшем предшествующем OPTION symbol. Структура OPTION завершается первым, следующим за ней #PROMPT, который не является RADIO.

Атрибут MULTI указывает, что программист может вводить в #PROMPT несколько значений. На окне свойств появляется кнопка с надписью description. Как вариант может быть указан атрибут ICON для поименования файла .ICO с выводимой пиктограммой. Эта кнопка вызывает окно, содержащая списковую структуру, показывающая все множество значений, введенных в #PROMPT с помощью кнопок Insert, Change и Delete. Эти три кнопки вызывают другое окно, содержащее #PROMPT string, чтобы позволить программисту редактировать элементы списка.

Когда программист вводит значение в #PROMPT, оно записывается в symbol. Введенное программистом значение может быть проверено на допустимость одним или более оператором #VALIDATE, следующими немедленно за оператором #PROMPT.

Размещенные в symbol значение(я) могут использоваться или оцениваться в шаблоне где угодно. Определенный в операторе #PROMPT в секции шаблона #APPLICATION symbol является глобальным, он может использоваться или оцениваться в шаблоне где угодно. Символ, определенный оператором #PROMPT в секции #PROCEDURE, является локальным и зависимым символом по отношению к %Procedure; он может использоваться или оцениваться только внутри этой секции #PROCEDURE. Символ, определенный оператором #PROMPT в секциях #CODE, #CONTROL или #EXTENSION, может использоваться или

оцениваться только внутри этих секций.

Пример:

```
#PROMPT ('Ask for Input',@s20),%InputSymbol      #!Simple input
#PROMPT ('Ask for FileName',FILE),%InputFile,REQ  #!Required filename
#PROMPT ('Pick One',OPTION),%InputChoice         #!Mutually exclusive
options
#PROMPT ('Choice One',RADIO)
#PROMPT ('Choice Two',RADIO)
#PROMPT ('Next Procedure',PROCEDURE),%NextProc   #!Prompt for procedure name
#PROMPT ('Ask for Multiple Input',@s20),%MultiSymbol,MULTI ('Input Values...')
                                                    #!Prompt for multiple input
См. также:  #DISPLAY,#VALIDATE,#GROUP,#BOXED,#ENABLE,#BUTTON
```

#VALIDATE (проверка ввода)

#VALIDATE (*expression, message*)

#VALIDATE	Проверяет данные, введенные в непосредственно предшествующем операторе #PROMPT.
<i>expression</i>	Выражение, которое используется для проверки введенных данных.
<i>message</i>	Строковая константа, содержащая сообщение об ошибке, которое выводится, если данные ошибочны.

Оператор #VALIDATE проверяет данные введенные в поле #PROMPT, который непосредственно предшествует оператору #VALIDATE. Выражение оценивается при нажатии кнопки ОК на окне Procedure Properties. Если выражение ложно, в окне сообщений выводится сообщение программисту и управление передается в поле #PROMPT, которое непосредственно предшествует оператору #VALIDATE. За #PROMPT может следовать несколько операторов #VALIDATE для проверки ввода.

Пример:

```
#PROMPT ('Input Value, Even numbers from 100-200',@N3),%Value
#VALIDATE ((%Value > 100) AND (%Value < 200),'Value must be between
100 and 200')
#VALIDATE ((%Value % 2 = 0),'Value must be an even number')
#PROMPT ('Screen Field',WINDOWCONTROL),%SomeField
#VALIDATE (%ScreenFieldType = 'LIST','Must select a list box')
См. также:  #PROMPT
```

#ENABLE (включение/выключение элементов диалога)

```
#ENABLE ( expression ) [, CLEAR ]
      prompts
#ENDENABLE
```

#ENABLE Начинает группу *prompts*, которая может быть включена или выключена в зависимости от результатов оценки *expression*.

expression Выражение, которое управляет включением / выключением элементов диалога.

CLEAR Указывает, что значение символов *prompts* должны быть очищены при выключении.

prompts Один или более операторов #PROMPT, #BUTTON, #DISPLAY, #ENABLE и/или #VALIDATE.

#ENDENABLE Заканчивает группу *prompts*.

Структура #ENABLE содержит *prompts*, которые могут быть включены или выключены в зависимости от результатов оценки *expression*. Если выражение истинно, *prompts* включаются, иначе выключаются. Когда *prompts* выключены, они выглядят полустертыми и программист не может вводить в них данные.

Пример:

```
#PROMPT ( 'Pick One',OPTION),%InputChoice          #!Mutually exclusive
options
#PROMPT ( 'Choice One',RADIO)
#PROMPT ( 'Choice Two',RADIO)
#ENABLE ( %InputChoice = 'Choice Two')
  #PROMPT ( 'Screen Field',WINDOWCONTROL),%SomeField  #!Enabled only
for Choice Two
  #VALIDATE ( %ScreenFieldType = 'LIST','Must select a list box')
#ENDENABLE
См. также:  #PROMPT,#GROUP,#BOXED,#BUTTON
```

#BUTTON (вызов новой страницы диалога)

```
#BUTTON ( string [, icon ] ) [,HLP ( id ) ] [, AT ( ) ] [, REQ ] [, INLINE ]
[, WHERE( condition ) ]
      [, |MULTI ( multisymbol, expression )      | ]
      |FROM ( fromsymbol, expression ) |
      prompts
#ENDBUTTON
```


#BUTTON	Создает командную кнопку для вызова новой страницы с prompts.
<i>string</i>	Строковая константа, содержащая надпись на кнопке. Она может содержать амперсанд (&), чтобы обозначить “горячую” клавишу.
<i>icon</i>	Строковая константа, содержащая спецификацию файла .ICO или стандартную пиктограмму кнопки. В этом случае string служит только для назначения “горячей” клавиши.
HLP	Указывает, что для #BUTTON доступна справка.
<i>id</i>	Строковая константа, содержащая идентификатор для обращения в справочную систему. Это может быть ключевым словом Справки или “строка контекста”
AT	Указывает положение группы в окне относительно первого размещенного шаблоном диалогового элемента (за исключением стандартных элементов в окне свойств процедуры). Этот атрибут имеет те же самые параметры, что и атрибут AT в языке Clarion.
REQ	Указывает, что программист должен по крайней мере один раз нажать кнопку при разработке процедуры.
MULTI	Указывает, что программист может ввести несколько наборов значений в элементы диалога. Это позволяет иметь многозначные связанные группы элементов.
<i>multisymbol</i>	Определяемый пользователем символ, от которого зависят все символы prompts. Это символ получает внутреннее уникальное значение для набора prompts.
INLINE	Указывает, что диалог имеет вид списка с кнопками редактирования, что позволяет пользователю ввести множество значений символа. Должен быть указан один из атрибутов MULTI или FROM.
<i>expression</i>	Строковое выражение для форматирования выводимых данных в списке значений многозначных символов.
FROM	Определяет, что программист может ввести в prompts набор значений для каждого экземпляра fromsymbol.
<i>fromsymbol</i>	Встроенный многозначный символ, который предопределяет все экземпляры, от которых зависят символы в prompts. Программист не может добавлять, удалять или менять экземпляры fromsymbol.
WHERE	Определяет, что #BUTTON выводит только те экземпляры fromsymbol для которых condition истинно.
<i>condition</i>	Выражение, которое задает условие применения.
<i>prompts</i>	Один или более операторов #PROMPT. Здесь могут применяться также операторы #DISPLAY, #VALIDATE, #ENABLE и #BUTTON.
#ENDBUTTON	Заканчивает группу prompts на странице, вызванной с помощью #BUTTON.

Оператор #BUTTON создает кнопку либо с string, либо с icon. Когда программист нажимает кнопку, возникает новая страница prompts.

Каждая новая страница prompts имеет свои кнопки OK, CANCEL и TEMPLATE HELP

в качестве стандартных полей. Все остальные поля на странице создаются с помощью prompts в структуре #BUTTON.

Кнопка ОК закрывает текущую станицу prompts, сохраняя при возврате в предыдущее окно данные, введенные программистом в prompts. Кнопка CANCEL закрывает текущее окно prompts без сохранения, затем возвращается в предыдущее окно. Если страница вызывает другую страницу с помощью вложенного оператора #BUTTON, и программист нажимает ОК на странице самого нижнего уровня, а затем CANCEL на странице, с которой она была вызвана, отменяется вся транзакция.

Атрибут MULTI указывает, что программист может ввести в prompts несколько наборов значений. Кнопка вызывает окно, содержащее списковую структуру, содержащую все множество значений, введенных для набора prompts и кнопки Insert, Change и Delete. Эти три кнопки вызывают другое окно, содержащие все prompts, чтобы позволить программисту изменять вводные поля списка. Для форматирования выводимых в списке значений используется expression.

Атрибут FROM также указывает, что программист может ввести в prompts несколько наборов значений. Кнопка вызывает окно, содержащее списковую структуру, содержащую все экземпляры fromsymbol вместе с кнопкой Edit. Кнопка вызывает другое окно, содержащие все prompts, чтобы позволить программисту изменять вводные поля, связанные с каждым из экземпляров fromsymbol.

Пример:

```
#PROMPT ('Name a File',FILE),%FileName          #!Prompt on the first page

#BUTTON ('Page Two')          #!Button on first page calls
#PROMPT ('Pick One',OPTION),%InputChoice        #!These prompts on second
page
#PROMPT ('Choice One',RADIO)
#PROMPT ('Choice Two',RADIO)
#ENABLE (%InputChoice = 'Choice Two')
#PROMPT ('Screen Field',WINDOWCONTROL),%SomeField
#VALIDATE (%ScreenFieldType = 'LIST','Must select a list box')
#ENDENABLE
#ENDBUTTON                                #!Terminate second page prompts

#PROMPT ('Enter some value',@S20),%InputValue1 #!Another prompt on the first
page

                                #!Multiple input button:
#BUTTON ('Multiple Names'),MULTI (%ButtonSymbol,%ForeName & ' ' & %SurName)
#PROMPT ('First Name',@S20),%ForeName
```

```
#PROMPT ('Last Name',@S20),%SurName
#ENDBUTTON                                #!Terminate second page prompts
```

```
#PROMPT ('Enter another value',@S20),%InputValue2 #!Another prompt on the first
page
```

```
#!Multiple input button dependent on %File:
```

```
#BUTTON ('File Options'),FROM (%File)
```

```
#PROMPT ('Open Access Mode',DROP ('Open|Share'),%FileOpenMode
```

```
#ENDBUTTON                                #!Terminate second page prompts
```

См. также: #PROMPT, #VALIDATE, #ENABLE

#FIELD (ввод для диалоговых элементов окна)

```
#FIELD, WHERE ( expression )
```

prompts

```
#ENDFIELD
```

#FIELD	Начинает группу элементов ввода для размещенных в окне элементов диалога.
WHERE	Указывает, что #FIELD используется только для тех экземпляров, когда <i>expression</i> истинно.
<i>expression</i>	Выражение, которое определяет условия использования.
<i>prompts</i>	Операторы размещения диалоговых элементов (#PROMPT, #BUTTON, и т.д.).
#ENDFIELD	Заканчивает группу.

Структура #FIELD содержит prompts для размещенных в окне диалоговых элементов. Эти prompts возникают в диалоге Actions... .

Список возникающих для поля элементов ввода строится в диалоге Actions... следующим образом:

1. Элементы ввода #CONTROL.
2. Элементы ввода #FIELD (а также из вложенных #GROUP) уровня #PROCEDURE.
3. Элементы ввода #FIELD из активных #EXTENSION уровня #PROCEDURE.
4. Элементы ввода #FIELD уровня #CONTROL.
5. Элементы ввода #FIELD уровня #CODE.

Значения, которые пользователь вводит в элементы ввода #FIELD, используются при генерации исходного текста, управляющего поведением диалогового элемента.

Пример:

```
#FIELD, WHERE (%ControlType = 'BUTTON')
#PROMPT ('Enter procedure call',PROCEDURE),%ButtonProc
#ENDFIELD
```

#PREPARE (подготовить символы приглашений)

```
#PREPARE
```

```
statements
```

```
#ENDPREPARE
```

#PREPARE

statements

Начинает блок подготовки символов приглашений

Операторы языка шаблонов, фиксирующие в многозначные символах те значения, которые необходимы для обработки последующих операторов #PROMPT или #BUTTON.

#ENDPREPARE

Завершает блок подготовки.

Структура #PRAPARE содержит операторы языка шаблонов, фиксирующие в многозначные символах те значения, которые необходимы для обработки последующих операторов #PROMPT или #BUTTON.

Пример:

```
#BUTTON('Customize Colors'),FROM(%ControlField,%ControlField), |
WHERE(%CntrlHasColor)
#PREPARE
#FIND(%ControlInstance,%ActiveTemplateInstance,%Control)
#ENDPREPARE
#BOXED('Default Colors')
#PROMPT('&Fore Normal:',COLOR),%ControlFieldForeNormal,DEFAULT(-
1)
#PROMPT('&Back Normal:',COLOR),%ControlFieldBackNormal,DEFAULT(-1)
#PROMPT('&Fore Selected:',COLOR),%ControlFieldForeSelected, |
DEFAULT(-1)
#PROMPT('&Back Selected:',COLOR),%ControlFieldBackSelected, |
DEFAULT(-1)
#ENDBOXED
#BOXED('Conditional Color Assignments')
#BUTTON('Conditional Colors'),MULTI(%ConditionalColors, |
%ColorCondition),INLINE
#PROMPT('&Condition:',@S255),%ColorCondition
#PROMPT('&Fore Normal:',COLOR),%CondControlFieldForeNormal, |
DEFAULT(-1)
#PROMPT('&Back Normal:',COLOR),%CondControlFieldBackNormal, |
DEFAULT(-1)
#PROMPT('&Fore Selected:',COLOR),%CondControlFieldForeSelected,|
DEFAULT(-1)
```

```
#PROMPT('&Back Selected:',COLOR),%CondControlFieldBackSelected,|
    DEFAULT(-1)
#ENDBUTTON
#ENDBOXED
#ENDBUTTON
```

Варианты type в #PROMPT

CHECK (переключатель)

CHECK

Значение CHECK в type оператора #PROMPT указывает, что symbol является переключателем, который может принимать значения только вкл/выкл, да/нет или истина/ложь. CHECK размещает на экране в области ввода #PROMPT переключатель. Когда переключатель включен, symbol диалогового элемента содержит единицу (1). Когда переключатель выключен, symbol диалогового элемента содержит ноль (0).

Пример:

```
#PROMPT ('Network Application',CHECK),%NetworkApp
```

COMPONENT (список полей ключа)

COMPONENT [(scope)]

COMPONENT

scope

Выводит список компонент ключа.

Символ, содержащий ключ. Если опущено, выводится список всех компонент всех ключей всех файлов.

Значение COMPONENT в type оператора #PROMPT указывает, что symbol должен содержать метку одного из полей, составляющих ключ. Когда очередь на экране свойств доходит до #PROMPT, возникает список доступных полей ключа.

Атрибут COMPONENT может иметь параметр *scope*, который ограничивает появляющиеся в списке компоненты ключа. Если *scope* является меткой ключа, список показывает все компоненты этого ключа.

Пример:

```
#PROMPT ('Record Selector',COMPONENT (%Primary)),%RecordSelector
```

CONTROL (список полей окна)

CONTROL

Значение CONTROL в type оператора #PROMPT указывает, что symbol должен содержать присвоенные полю метки для диалоговых элементов окна. Когда очередь на экране свойств доходит до #PROMPT, возникает список доступных диалоговых элементов.

Пример:

```
#PROMPT ('Locator Field',CONTROL),%Locator
```

DROP (выпадающий список)

DROP [(*scope*)]

DROP

Создает выпадающий список значений.

scope

Строковая константа, содержащая разделенные вертикальной чертой (|) значения для построения списка.

Значение DROP в type оператора #PROMPT указывает, что symbol должен содержать один из элементов списка, заданного параметром *scope*.

Для каждого варианта *scope* может дополнительно содержать в квадратных скобках текст, который будет присвоен указанному в #PROMPT символу. Это используется при переводах.

Параметр *scope* должен содержать все элементы списка. Список значений выпадает аналогично диалоговому элементу LIST языка Clarion с атрибутом DROP. Если не указано стандартное значение, symbol принимает значение первого элемента списка *scope*.

Пример:

```
#PROMPT ('If file not found',DROP ('Create the file|Halt Program')),%FileNotFound
```

```
#PROMPT('Что это?',DROP('Имя[Name]|Число[Number]')),%WhatIsThis
```

```
!# При выборе "Имя", %WhatIsThis будет присвоено "Name"
```

```
!# При выборе "Число", %WhatIsThis будет присвоено "Number"
```

EMBED (ввод текста в точку вставки)

EMBED (*identifier* [, *instance*])

EMBED

Указывает, что элемент диалога непосредственно редактирует текст в точке вставки.

identifier

Определяемый пользователем символ, который указывает на точку

instance вставки #EMBED, в которой производится редактирование. Строковая константа или выражение, содержащее одно из значений использованного в #EMBED многозначного символа. Вы можете указать столько instances сколько необходимо для точной идентификации редактируемого экземпляра точки вставки #EMBED.

Значение EMBED в type оператора #PROMPT указывает, что symbol используется для непосредственного редактирования в точке вставки исходного текста. Это размещает область ввода и кнопку с многоточием (...) рядом с элементом диалога, чтобы позволить пользователю иметь доступ к точке вставки исходного текста. Программист может ввести вызов процедуры в область ввода или нажать кнопку с многоточием для перехода к обычному вводу исходного текста.

Если #EMBED связан с многозначным символом, нужно указать определенный экземпляр (instance) #EMBED. При использовании многозначного символа в качестве выражения для instance, он должен быть фиксирован на единственном значении. Вообще говоря, это должно использоваться в структуре #FIELD.

Пример:

```
#PROMPT ('Embedded Data Declarations',EMBED (%DataSection))
#FIELD, WHERE (%ControlType = 'BUTTON')
    #PROMPT ('Action when button is pressed',EMBED
(%ControlEvent,%Control,'Accepted'))
#ENDFIELD
```

EMBEDBUTTON (ввод текста в точку вставки)

EMBEDBUTTON (*identifier* [, *instance*])

EMBEDBUTTON Указывает, что элемент диалога непосредственно редактирует текст в точке вставке.

identifier Определяемый пользователем символ, который указывает на точку вставки #EMBED, в которой производится редактирование.

instance Строковая константа или выражение, содержащее одно из значений использованного в #EMBED многозначного символа. Вы можете указать столько instances, сколько необходимо для точной идентификации редактируемого экземпляра точки вставки #EMBED.

Значение EMBEDBUTTON в type оператора #PROMPT указывает, что symbol используется для непосредственного редактирования в точке вставки исходного текста. Это размещает рядом с приглашением кнопку, позволяющую пользователю иметь доступ к точке вставки исходного текста. Программист может нажать кнопку для перехода к

вводу исходного текста.

Если `#EMBED` связан с многозначным символом, нужно указать определенный экземпляр (instance) `#EMBED`. При использовании многозначного символа в качестве выражения для instance, он должен быть фиксирован на единственном значении. Вообще говоря, это должно использоваться в структуре `#FIELD`.

Пример:

```
#PROMPT ('Embedded Data Declarations',EMBEDBUTTON (%DataSection))
```

```
#FIELD, WHERE (%ControlType = 'BUTTON')
      #PROMPT ('Action for button press',EMBEDBUTTON
(%ControlEvent,%Control,'Accepted'))
#ENDFIELD
```

FIELD (список полей данных)

FIELD [(*scope*)]

FIELD

scope

Выводит список полей в файлах.

Символ, содержащий метку файла. Если опущен, то выводятся все поля всех файлов.

Значение `FIELD` в type оператора `#PROMPT` указывает, что `symbol` должен содержать метку поля файла данных. Когда очередь на экране свойств доходит до `#PROMPT`, возникает список доступных полей.

Параметр `scope` может применяться для ограничения доступных в списке полей. Если `scope` указывает на файл, в списке выводятся все поля файла. Если параметр `scope` не указан, в списке выводятся все поля всех файлов.

Пример:

```
#PROMPT ('Locator Field',FIELD (%Primary)),%Locator
```

FILE (список файлов)

FILE

Значение `FILE` в type оператора `#PROMPT` указывает, что `symbol` должен содержать метку файла данных. Когда очередь на экране свойств доходит до `#PROMPT`, возникает схема файлов процедуры.

Пример:

#PROMPT ('Logout File',FILE),%LogoutFile

FORMAT (вызов форматера списковой структуры)

FORMAT

Значение FORMAT в type оператора #PROMPT указывает, что symbol должен содержать атрибут FORMAT спискового диалогового элемента LIST, поэтому он вызывает для его построения форматер списковой структуры.

Пример:

#PROMPT ('Alternate LIST format',FORMAT),%AlternateFormatString

FROM (список значений символа)

FROM (*symbol* [, *expression*] [, *value*])

FROM	Определяет выпадающий список значений <i>symbol</i> .
<i>symbol</i>	Многозначный символ.
<i>expression</i>	Выражение, которое определяет, какие из значений <i>value</i> включаются в список. Только те значения <i>value</i> , для которых <i>expression</i> истинно, выводятся в выпадающем списке.
<i>value</i>	Символ, содержащий выводимые в элементе диалога значения, которые присваиваются символу <i>symbol</i> .

Значение FROM в type оператора #PROMPT указывает, что пользователь должен выбрать одно значение из списка, содержащегося в *value*. Для ограничения выводимых значений *value* используется *expression*, в то время, как *value* определяет выводимые элементы.

Пример:

```
#PROMPT ('Select an Event',FROM (%ControlEvent)),%WhichEvent
#PROMPT ('Select a Button',FROM (%ControlField,%ControlType =
'BUTTON')),%WhichButton
#PROMPT ('Pick a Field',FROM (%Control,%ControlUse <>
',,%ControlUse)),%MyButton
```

KEY (список ключей)

KEY [(*scope*)]

KEY	Выводит список ключей.
<i>scope</i>	Символ, определяющий файл. Если опущен, список выводит все ключи всех файлов.

Значение KEY в type оператора #PROMPT указывает, что symbol должен содержать метку ключа файла. Когда очередь на экране свойств доходит до #PROMPT, возникает список ключей из словаря данных.

Параметр score может применяться для ограничения доступных в списке ключей. Если score указывает на файл, в списке выводятся все ключи файла. Если параметр score не указан, в списке выводятся все ключи всех файлов.

Пример:

```
#PROMPT ('Which Key',KEY,%Primary),%UseKey
```

KEYCODE (список кодов клавиш)

KEYCODE

Значение KEYCODE в type оператора #PROMPT указывает, что symbol должен содержать код клавиши или символическую метку кода клавиши. Когда очередь на экране свойств доходит до #PROMPT, возникает список символических меток кодов клавиш из файла KEYCODES.EQU.

Пример:

```
#PROMPT ('Hot Key',KEYCODE),%ActiveKey
```

OPENDIALOG (вызвать диалог Open File)

OPENDIALOG(*title*, *extensions*)

OPENDIALOG	Вызывает стандартный диалог Windows Open File.
<i>title</i>	Строковая константа, содержащая заголовок диалога Open File.
<i>extensions</i>	Строковая константа, содержащая возможные варианты выбора файла для выпадающего списка типов файлов.

Значение PICTURE в type оператора #PROMPT указывает, что вводимый символ должен содержать имя файла, которое программист выбирает с помощью диалога Open File. Файл должен уже существовать на диске.

Пример:

```
#PROMPT('Ask for File',OPENDIALOG('Pick File', 'Source|*.CLW')), |
%FileSymbol
```

OPTION (вывод радиокнопок)

OPTION

Значение OPTION в type оператора #PROMPT указывает, что symbol должен содержать значение string одного из последующих операторов RADIO #PROMPT, если только не

присутствует атрибут CHOICE. В этом случае символ вместо строки получает значение порядкового номера выбранного в группе OPTION оператора RADIO #PROMPT.

Пример:

```
#PROMPT ('Ask for Choice',OPTION),%OptionSymbol  
#PROMPT ('Option One',RADIO)  
#PROMPT ('Option Two',RADIO)  
#PROMPT ('Option Three',RADIO)
```

PICTURE (вызов формatera шаблонов ввода)

PICTURE

Значение PICTURE в type оператора #PROMPT вызывает формater шаблонов ввода для создания шаблона, используемого при выводе данных на экран.

Пример:

```
#PROMPT('Display Format',PICTURE),%DisplayPicture
```

PROCEDURE (добавление процедуры к дереву процедур)

PROCEDURE

Значение PROCEDURE в type оператора #PROMPT указывает, что значение, находящиеся в symbol, является именем процедуры приложения. String выбранного RADIO записывается в symbol оператора #PROMPT с атрибутом OPTION, если только не присутствует атрибут CHOICE. В этом случае символ вместо строки получает значение порядкового номера выбранного в группе OPTION оператора RADIO #PROMPT.

Пример:

```
#PROMPT ('Next Procedure',PROCEDURE),%NextProcedure
```

RADIO (единичная радиокнопка)

RADIO

Значение RADIO в type оператора #PROMPT создает одну радиокнопку для ближайшего предшествующего диалогового элемента OPTION. String выбранного RADIO записывается в symbol оператора #PROMPT с атрибутом OPTION.

Пример:

```
#PROMPT ('Ask for Choice',OPTION),%OptionSymbol
```

```
#PROMPT ('Option One',RADIO)
#PROMPT ('Option Two',RADIO)
#PROMPT ('Option Three',RADIO)
```

```
#PROMPT('Ask for Another Choice',OPTION),%OptionSymbol2,CHOICE
#PROMPT('Option One',RADIO)      #! %OptionSymbol2 будет присвоено 1
#PROMPT('Option Two',RADIO)      #! %OptionSymbol2 будет присвоено 2
#PROMPT('Option Three',RADIO)    #! %OptionSymbol2 будет присвоено 3
```

SAVEDIALOG (вызвать диалог Save File)

SAVEDIALOG(*title, extensions*)

SAVEDIALOG	Вызывает стандартный диалог Windows Save File.
<i>title</i>	Строковая константа, содержащая заголовок диалога Save File.
<i>extensions</i>	Строковая константа, содержащая возможные варианты выбора файла для выпадающего списка типов файлов.

Значение PICTURE в type оператора #PROMPT указывает, что вводимый символ должен содержать имя файла, которое программист выбирает с помощью диалога Save File. Файл должен уже существовать на диске.

Пример:

```
#PROMPT('Ask for FileName',SAVEDIALOG('Pick File', 'Source|*.CLW')), |
%FileSymbol
```

SPIN (элемент диалога с “прокруткой”)

SPIN(*picture, low, high* [, *step*])

SPIN	Создает диалоговый элемент с “прокруткой”.
<i>picture</i>	Шаблон форматирования вводных данных.
<i>low</i>	Числовая константа или выражение, содержащая наименьшее допустимое значение.
<i>high</i>	Числовая константа или выражение, содержащая наибольшее допустимое значение.
<i>step</i>	Числовая константа или выражение, содержащая величину приращения при каждом изменении от наименьшего к наибольшему допустимому значению. Если опущено, используется 1.

Значение SPIN в type оператора #PROMPT создает элемент диалога с “прокруткой” для того, чтобы программист мог ввести допустимое число.

Пример:

```
#PROMPT ('How Many?',SPIN(@n2,1,10)),%SpinSymbol
```

Операторы вывода и оформления.

#BOXED (группа элементов диалога)

```
#BOXED ( string ) [ , AT ( ) ] [ , WHERE ( expression ) ] [ , CLEAR ] [ , HIDE ]
      prompts
#ENDBOXED
```

#BOXED	Создает группу prompts.
<i>string</i>	Строковая константа, содержащая текст, выводимый в заголовке группы.
AT	Указывает положение группы в окне относительно первого размещенного шаблоном диалогового элемента (за исключением стандартных элементов в окне свойств процедуры). Этот атрибут имеет те же самые параметры, что и атрибут AT в языке Clarion.
WHERE	Указывает, что #BOXED доступен только для тех экземпляров, в которых <i>expression</i> истинно.
<i>expression</i>	Выражение, задающее условие использования.
CLEAR	Указывает, что значение символа prompts очищаются при выключении.
<i>prompts</i>	Один или более операторов #PROMPT. Здесь могут применяться также операторы #DISPLAY, #VALIDATE, #ENABLE и #BUTTON.
HIDE	Указывает, что prompts скрыты, если выражение WHERE ложно при первом появлении диалога.
#ENDBOXED	Заканчивает группу диалоговых элементов.

Оператор #BOXED создает группу диалоговых элементов с *string* в качестве заголовка. Если присутствует атрибут WHERE, prompts будут скрыты или видимы, в зависимости от результатов оценки *expression*. Если *expression* истинно, prompts видимы, иначе - скрыты.

Пример:

```
#PROMPT ('Pick One',OPTION),%InputChoice  #!These prompts on second page
#PROMPT ('Choice One',RADIO)
#PROMPT ('Choice Two',RADIO)
#BOXED ('Choice Two Options'),WHERE (%InputChoice = 'Choice Two')
    #PROMPT ('Screen Field',WINDOWCONTROL),%SomeField
    #VALIDATE (%ScreenFieldType = 'LIST','Must select a list box')
#ENDBOXED
```

См. также: #PROMPT, #VALIDATE

#DISPLAY (вывод)

#DISPLAY ([*string*]) [, AT()]

#DISPLAY Выводит строковую константу в окне свойств.
string Строковая константа, содержащая выводимый текст.
AT Указывает, положение и размеры области вывода *string*, позволяя выводить несколько строк. Этот атрибут имеет такие же параметры, как и атрибут **AT** в языке Clarion.

Оператор **#DISPLAY** выводит *string* в окне свойств. Если *string* опущена, выводится пустая строка. **#DISPLAY** используется в сочетании с **#PROMPT** для формирования многострочных приглашений или для создания пробелов на окне свойств. **#DISPLAY** не может применяться в секции **#MODULE**.

Пример:

```
#DISPLAY ()      #!Display a blank line
#DISPLAY ('Ask programmer to input some') #!Display a string
#PROMPT (' specific value',@s20),%InputSymbol
См. также:#PROMPT,#GROUP,#BOXED,#ENABLE,#BUTTON
```

#IMAGE (вывести графическое изображение)

#IMAGE(*picture*, [, AT()])

#IMAGE Выводит в диалоге свойств графическое изображение.
picture Строковая константа, содержащая имя файла с изображением.
AT Указывает, положение и размеры области вывода *picture*. Этот атрибут имеет такие же параметры, как и атрибут **AT** в языке Clarion.

Оператор **#IMAGE** выводит в окне графическое изображение *picture*. **#IMAGE** не может использоваться в секции **#MODULE**.

Пример:

```
#IMAGE('SomePic.BMP')      #!Display a bitmap
```

#SHEET (объявляет группу диалоговых элементов #TAB)

#SHEET

tabs

#ENDSHEET

#SHEET Объявляет группу диалоговых элементов **#TAB**.
tabs Объявления диалоговых элементов **#TAB**.
#ENDSHEET Заканчивает объявление группы диалоговых элементов **#TAB**.

Пример:

```
#UTILITY(ApplicationWizard,'Create a New Database Application'),WIZARD
#!
#SHEET
  #TAB('Application Wizard')
    #IMAGE('CMPAPP.BMP')
    #DISPLAY('This wizard will create a new Application. '), %|
AT(90,8,235,24)
    #DISPLAY('To begin creating your new Application, click Next. '),%|
AT(90)
  #ENDTAB
  #TAB('Application Wizard - File Usage'),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #DISPLAY('You can gen procs for all files, or select them '),%|
AT(90,8,235,24)
    #PROMPT('Use all files in DCT',CHECK),%GenAllFiles,AT(90,,180), %|
    DEFAULT(1)
  #ENDTAB
  #TAB('Select Files to Use'),WHERE(NOT %GenAllFiles),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #PROMPT('File Select',FROM(%File)),%FileSelect,INLINE, %|
    SELECTION('File Select')
  #ENDTAB
  #TAB('Application Wizard - Finally...'),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #DISPLAY('Old procs can be overwritten or new procs suppressed')
    #PROMPT('Overwrite existing procs',CHECK),%OverwriteAll, %|
    AT(90,,235),DEFAULT(0)
    #IMAGE('<255,1,4,127>'),AT(90,55)
    #DISPLAY('Your First Procedure is always overwritten! '), %|
AT(125,54,200,20)
  #ENDTAB
#ENDSHEET
```

#TAB (объявляет страницу диалогового элемента #SHEET)

```
#TAB( text ) [, FINISH( expression ) ] [, WHERE( expression ) ]
    prompts
#ENDTAB
```

#TAB	Объявляет группу приглашений prompts, которые образуют одну из страниц в структуре #SHEET.
text	Строковая константа, содержащая текст закладки к странице или текст

	заголовка окна диалога, если используется в секции #UTILITY с атрибутом WISARD.
FINISH	Указывает наличие кнопки “Finish”. Используется только в секции #UTILITY с атрибутом WISARD.
<i>expression</i>	Выражение, которое задает условие использования.
WHERE	Указывает, что страничка видна только при выполнении условия expression.
<i>expression</i>	Выражение, которое задает условие использования.
<i>prompts</i>	Один или более операторов #PROMPT. Здесь могут также использоваться операторы #DISPLAY, #VALIDATE, #ENABLE, и #BUTTON
#ENDTAB	Заканчивает группу приглашений prompts.

Структура #TAB объявляет группу приглашений prompts, которые образуют одну из страниц в структуре #SHEET. Несколько элементов #TAB в структуре #SHEET определяют страницы пользовательского диалога.

Пример:

```
#UTILITY(ApplicationWizard,'Create a New Database Application'),WIZARD
#!
#SHEET
  #TAB('Application Wizard')
    #IMAGE('CMPAPP.BMP')
    #DISPLAY('This wizard will create a new Application. '), |
AT(90,8,235,24)
    #DISPLAY('To begin creating your new Application, click Next. '), |
AT(90)
  #ENDTAB
  #TAB('Application Wizard - File Usage'),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #DISPLAY('You can gen procs for all files, or select them '), |
AT(90,8,235,24)
    #PROMPT('Use all files in DCT',CHECK),%GenAllFiles,AT(90,,180), |
    DEFAULT(1)
  #ENDTAB
  #TAB('Select Files to Use'),WHERE(NOT %GenAllFiles),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #PROMPT('File Select',FROM(%File)),%FileSelect,INLINE, |
    SELECTION('File Select')
  #ENDTAB
  #TAB('Application Wizard - Finally...'),FINISH(1)
    #IMAGE('WINAPP.BMP')
    #DISPLAY('Old procs can be overwritten or new procs suppressed')
    #PROMPT('Overwrite existing procs',CHECK),%OverwriteAll, |
```



```
        AT(90,,235),DEFAULT(0)
        #IMAGE('<255,1,4,127>'),AT(90,55)
        #DISPLAY('Your First Procedure is always overwritten!'), |
AT(125,54,200,20)
        #ENDTAB
        #ENDSHEET
```


Управление генерацией текста

Операторы управления логикой

#FOR (многократная генерация текста)
#IF (условная генерация текста)
#LOOP (итеративная генерация текста)
#CASE (структура условного выполнения)
#INSERT (вставить текст из #GROUP)
#BREAK (выход из цикла)
#CYCLE (переход в начало цикла)
#RETURN (возврат из #GROUP)
#GENERATE (генерация секции исходного текста)
#ABORT (прекращение генерации)

Операторы управления файлами

#CREATE (создание файла исходного текста)
#OPEN (открыть файл исходного текста)
#CLOSE (закрыть файл исходного текста)
#READ(чтение одной строки из файла исходного текста)
#REDIRECT (изменить файл исходного текста)
#APPEND (добавить в конец файла исходного текста)
#REMOVE (удаление файла исходного текста)
#REPLACE (условная замена файла исходного текста)
#PRINT (печать файла исходного текста)

Операторы условной генерации

#SUSPEND (начать условный текст)
#RELEASE (подтверждение условной генерации текста)
#RESUME (граница условной генерации текста)
#? (условная строка условного текста)

Операторы управления логикой

#FOR (многократная генерация текста)

```
#FOR ( symbol ) [, WHERE ( expression ) ] [, REVERSE ]
      statements
#ENDFOR
```

#FOR	Цикл по всем экземплярам многозначного символа.
<i>symbol</i>	Многозначный символ.
WHERE	Указывает, что <i>statements</i> в цикле #FOR исполняются только для тех экземпляров <i>symbol</i> , для которых истинно <i>expression</i> .
<i>expression</i>	Выражение, которое задает условие исполнения.
REVERSE	Указывает, что цикл #FOR проходит по экземплярам символа в обратном порядке.
<i>statements</i>	Операторы языка шаблонов и целевого языка.
#ENDFOR	Завершает структуру #FOR.

Цикл #FOR определяет циклическую структуру, которая во время генерации текста генерирует *statements* по одному разу для каждого значения *symbol*. Если в *symbol* не содержится значений, то не генерируется никакого текста. Оператор #FOR должен завершаться оператором #ENDFOR. Если #ENDFOR отсутствует, то во время препроцессирования шаблона выдается сообщение об ошибке. Цикл #FOR может быть вложенным в другой цикл #FOR.

Цикл #FOR начинает с первого экземпляра *symbol* (или с последнего, если указан атрибут REVERSE) и обрабатывает все экземпляры *symbol*, если не подвергается воздействию оператора #FIX. Если указан атрибут WHERE, операторы *statements* в цикле #FOR выполняются только для тех экземпляров *symbol*, для которых истинно *expression*. Это образует условный цикл #FOR.

Так как #FOR является структурой цикла, для управления циклом могут применяться операторы #BREAK и #CYCLE. Оператор #BREAK немедленно прекращает обработку в цикле #FOR, и продолжает ее с первого оператора, следующего за тем #ENDFOR, который завершает этот цикл. Оператор #CYCLE немедленно передает управление оператору #FOR, для того, чтобы продолжить работу с новым экземпляром *symbol*.

Пример:

```
#FOR (%ScreenField),WHERE (%ScreenFieldType = 'LIST')
  #INSERT (%ListQueueBuild)           #!Generate only for LIST controls
#ENDFOR
```

См. также: #BREAK, #CYCLE

#IF (условная генерация текста)

```
#IF( expression )  
    statements  
[ #ELSIF( expression )  
    statements ]  
[ #ELSE  
    statements ]  
#ENDIF
```

#IF <i>expression</i>	Структура условного исполнения. Любое выражение на языке шаблонов, которое оценивается как ложь (пробел или ноль) или истина (любое другое значение). Выражение может содержать символы шаблона, константы и любые арифметические или логические операторы, описанные в Language Reference. Разрешаются вызовы функций. Если в выражении используется операция деления по модулю (%), она должна отделяться по крайней мере одним пробелом с каждой стороны (чтобы отличаться от символов шаблона).
<i>statements</i>	Один или более операторов языка шаблонов и/или Clarion.
#ELSIF	Задаёт альтернативное условие, которое оценивается, когда условия в предшествующих #IF и #ELSIF оказались ложны.
#ELSE	Задаёт альтернативное условие, которое оценивается, когда условия в предшествующих #IF и #ELSIF оказались ложны.
#ENDIF	Завершает структуру #IF.

Оператор #IF выборочно генерирует одну из групп *statements* в зависимости от результатов оценки *expression*. Структура #IF состоит из оператора #IF и всех, следующих за ним операторов, пока не завершится оператором #ENDIF. Если #ENDIF отсутствует, то во время препроцессирования шаблона выдается сообщение об ошибке. Структура #IF может быть вложена в другую структуру #IF.

Операторы #ELSIF и #ELSE являются логическими разделителями, которые делят структуру #IF на группы *statements*, которые условно генерируются в зависимости от результатов оценки *expression*. В структуре #IF может быть несколько операторов #ELSIF, но только один #ELSE.

Если в процессе генерации текста встречается #IF:

- Если *expression* оценивается как истинное, генерируются только операторы, расположенные между #IF и последующим #ELSIF, #ELSE или #ENDIF.

- Если *expression* оценивается как ложное, тем же способом оценивается **#ELSIF** (если имеется). Если *expression* в **#ELSIF** истинно, генерируются только операторы, расположенные между этим **#ELSIF** и последующим **#ELSIF**, **#ELSE** или **#ENDIF**.
- Если все предыдущие условия в **#IF** и **#ELSIF** оцениваются как ложные, генерируются только операторы, расположенные между **#ELSE** (если имеется) и **#ENDIF**. Если **#ELSE** отсутствует, то не генерируется никакого текста.

Пример:

```
#IF (SUB(%ReportControlStatement,1,6)='HEADER')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='FOOTER')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='DETAIL')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='OPTION')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='GROUP')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='BREAK')
  #SET (%Indentation,%Indentation+1)
#ELSIF(SUB(%ReportControlStatement,1,6)='FORM')
  #SET (%Indentation,%Indentation+1)
#ENDIF
```

#LOOP (итеративная генерация текста)

```
#LOOP [ , | UNTIL ( expression ) | ]
      | WHILE ( expression ) |
      | FOR( counter, start, end ) [, BY( step ) ] |
      statements
#ENDLOOP
```

#LOOP UNTIL

Начинает структуру итеративного выполнения операторов. Оценивает *expression* перед каждой итерацией **#LOOP**. Если *expression* оценивается как истинное, управляющая структура **#LOOP** завершается.

expression

Любое выражение на языке шаблонов, которое оценивается как ложь (пробел или ноль) или истина (любое другое значение).

WHILE

Оценивает *expression* перед каждой итерацией **#LOOP**. Если *expression* оценивается как ложное, управляющая структура **#LOOP** завершается.

FOR

Инициализирует *counter* начальным значением *start* и увеличивает

<i>counter</i>	его на величину <i>step</i> при каждом повторении цикла. Когда <i>counter</i> становится больше, чем значение <i>end</i> , выполнение цикла прекращается.
<i>start</i>	Определяемый символ, используемый в качестве счетчика.
<i>end</i>	Выражение, содержащее начальное значение, в которое должен быть установлен <i>counter</i> .
BY	Выражение, содержащее конечное значение счетчика цикла <i>counter</i> .
<i>step</i>	Явно определяет величину приращения счетчика цикла <i>counter</i> .
<i>statements</i>	Выражение, содержащее величину приращения счетчика цикла <i>counter</i> . Если опущено, используется единица (1).
#ENDLOOP	Один или более операторов языка шаблонов и/или Clarion. Завершает структуру #LOOP .

Структура **#LOOP** повторяет выполнение входящих в нее операторов. Структура **#LOOP** должна завершаться оператором **#ENDLOOP**. Если **#ENDLOOP** отсутствует, то во время препроцессирования шаблона выдается сообщение об ошибке. Структура **#LOOP** может быть вложенной в другую структуру **#LOOP**.

Операторы **#LOOP**, **UNTIL** или **#LOOP, WHILE** содержат условия завершения цикла **#LOOP**. Выражения в них всегда оцениваются в начале цикла перед его выполнением. Структура **#LOOP, WHILE** непрерывно повторяется, пока условие истинно. Структура **#LOOP, UNTIL** непрерывно повторяется, пока условие ложно. Выражение может содержать символы шаблона, константы и любые арифметические или логические операторы описанные в Language Reference. Разрешаются вызовы функций. Если в выражении используется операция деления по модулю (%), она должна отделяться по крайней мере одним пробелом с каждой стороны (чтобы отличаться от символов шаблона).

Оператор **#LOOP, FOR** также явно задает условия окончания **#LOOP**. Оператор **#LOOP** инициализирует *counter* начальным значением *start* для первой итерации. Оператор **#LOOP** автоматически увеличивает *counter* на величину *step* при каждой последующей итерации, а затем сравнивает его с конечным значением *end*. Когда *counter* больше *end*, выполнение **#LOOP** прекращается.

Оператор **#LOOP** (без **WHILE**, **UNTIL** или **FOR**) выполняется непрерывно, пока не выполнится оператор **#BREAK** или **#RETURN**. Оператор **#BREAK** прекращает выполнение **#LOOP** и продолжает выполнение с первого оператора, следующего за структурой **#LOOP**. Операторы в структуре **#LOOP** выполняются, пока не выполнится оператор **#CYCLE**. Оператор **#CYCLE** немедленно передает управление обратно в начало цикла, чтобы начать следующую итерацию без выполнения операторов, следующих в структуре **#LOOP** за оператором **#CYCLE**.

Пример:

```
#SET (%LoopBreakFlag,'NO')
#LOOP      #!Continuous loop
```

```

#INSERT (%SomeRepeatedCodeGroup)
#IF (%LoopBreakFlag = 'YES') #!Check break condition
    #BREAK
#ENDIF
#ENDLOOP
#SET (%LoopBreakFlag, 'NO')
#LOOP, UNTIL (%LoopBreakFlag = 'YES') #!Loop until condition is true
    #INSERT (%SomeRepeatedCodeGroup)
#ENDLOOP
#SET (%LoopBreakFlag, 'NO')
#LOOP, WHILE (%LoopBreakFlag = 'NO') #!Loop while condition is true
    #INSERT (%SomeRepeatedCodeGroup)
#ENDLOOP
См. также: #BREAK, #CYCLE

```

#CASE (структура условного выполнения)

```

#CASE ( condition )
#OF ( expression )
[ #OROF ( expression ) ]
    statements
[ #ELSE
    statements ]
#ENDCASE

```

#CASE
condition

Начинает структуру выборочного выполнения операторов.
Любое выражение на языке шаблонов, которое возвращает значение.
#OF Операторы *statements* после **#OF** выполняются, когда *expression* в **#OF** равно *condition* в **#CASE**. В структуре **#CASE** может быть много вариантов **#OF**.

expression
#OROF

Любое выражение на языке шаблонов, которое возвращает значение.
Операторы *statements* после **#OROF** выполняются, когда *expression* в **#OROF** равно *condition* в **CASE**. С одним вариантом **#OF** может быть связано много вариантов **#OROF**.

#ELSE

Операторы *statements* после **#ELSE** выполняются, когда все предыдущие *expression* в **#OF** и **#OROF** не равны *condition* в **#CASE**. Вариант **#ELSE** (если используется) должен быть последним вариантом в структуре **#CASE**.

statements
#ENDCASE

Любой правильный исполняемый исходный текст.
Завершает структуру **#CASE**.

Структура **#CASE** выборочно выполняет операторы *statements* на основании сравнения

условия condition в #CASE и выражений expression в #OF или #OROF. Если нет точного соответствия, то выполняются операторы после #ELSE. Если #ENDCASE отсутствует, то во время препроцессирования шаблона выдается сообщение об ошибке. Структура #CASE может быть вложенной в другие структуры #CASE.

Пример:

```
#CASE %ScreenField
#OF 'Ok'
  #INSERT (%OkButtonGroup)
#OF 'Cancel'
#OROF 'Exit'
  #INSERT (%CancelButtonGroup)
#ELSE
  #INSERT (%OtherControlsGroup)
#ENDCASE
```

#INSERT (вставить текст из #GROUP)

#INSERT (*symbol* [(*set*)] [, *parameters*])

#INSERT	Вставляет текст из #GROUP.
<i>symbol</i>	Символ, определяющий имя секции #GROUP.
<i>set</i>	Имя набора шаблонов (параметр name оператора #TEMPLATE) которому принадлежит группа. Если опущено, #GROUP должна быть в наборе шаблонов с тем же именем, что и #PROCEDURE в которой она используется.
<i>parameters</i>	Передаваемые в #GROUP параметры. Каждый параметр должен быть отделен запятой. Должны быть переданы все параметры, определенные в группе; их нельзя опускать.

Оператор #INSERT размещает в том месте шаблона, в котором он находится, текст из секции #GROUP, указанной в его symbol. Параметр set указывает #TEMPLATE, содержащий #GROUP. Это позволяет в любом шаблоне использовать группы из любого зарегистрированного шаблона.

Передаваемые в #GROUP параметры распадаются на две категории: параметры-значения и параметры-переменные. Параметры-значения объявляются в #GROUP как определяемые символы с предшествующей звездочкой (*). В качестве параметра-значения могут передаваться как символы, так и выражения. В качестве параметра-переменной могут передаваться только символы.

Пример:

```

#INSERT (%SomeGroup)                #!Ordinary insert
#INSERT (%GenerateFormulas('Clarion')) #!Insert #GROUP from Clarion Tem-
plate
#INSERT (%FileRecordFilter,%Secondary)  #!Insert #GROUP with passed param-
eter
#INSERT (%FileRecordFilter(Clarion),%Primary,%Secondary)
    #!#GROUP from Clarion Template with two passed parameters
См. также: #GROUP

```

#BREAK (выход из цикла)

#BREAK

Оператор #BREAK немедленно выходит из той структуры #FOR или #LOOP, в которой он встречается. Управление передается следующему за #ENDFOR или #ENDLOOP оператору. Оператор #BREAK может находиться только внутри структуры #FOR или #LOOP, иначе во время препроцессирования шаблона выдается сообщение об ошибке.

Пример:

```

#SET (%StopFile,'CUSTOMER')
#FOR (%File)
    #IF (UPPER(%File) = %StopFile)
        #BREAK
    #ENDIF
    OPEN(%File)
#ENDFOR

```

#CYCLE (переход в начало цикла)

#CYCLE

Оператор #CYCLE немедленно передает управление в начало того цикла #FOR или #LOOP в котором он находится, начиная следующую итерацию. Оператор #CYCLE может находиться только внутри структуры #FOR или #LOOP, иначе во время препроцессирования шаблона выдается сообщение об ошибке.

Пример:

```

#SET (%StopFile,'CUSTOMER')
#FOR (%File)
    #IF (UPPER(%File) <> %StopFile)
        OPEN(%File)
    #CYCLE

```

```
#ELSE
  #BREAK
#ENDIF
#ENDFOR
```

#RETURN (возврат из #GROUP)

#RETURN

Оператор #RETURN немедленно возвращает управление на оператор, следующий за тем оператором #INSERT, который вызвал #GROUP, содержащую оператор #RETURN. #RETURN может находиться только в секции #GROUP.

Пример:

```
#GROUP (%ProcessListGroup,%PassedControl)
  #FIX (%ScreenField,%PassedControl)
  #IF (%ScreenFieldType <> 'LIST')
    #UNFIX (%ScreenField)
    #RETURN
  #ENDIF
```

#GENERATE (генерация секции исходного текста)

#GENERATE (*section*)

#GENERATE

section

Генерирует секцию приложения.

Один из следующих встроенных символов: %Program, %Module или %Procedure. Этот символ определяет генерируемую часть приложения.

Оператор #GENERATE генерирует исходный текст для указанной секции *section* приложения, выполняя операторы языка шаблонов, содержащиеся в этой *section*. Оператор #GENERATE должен использоваться только в секциях #APPLICATION или #UTILITY шаблона.

Для *section*:

%Program

Генерируется #PROGRAM секция шаблона.

%Module

Генерируется соответствующая #MODULE секция шаблона.

%Procedure

Генерируется #PROCEDURE секция шаблона для соответствующей %Procedure.

Пример:

#GENERATE (%Program)	#!Generate program
header	
#FOR (%Module)	#!
#GENERATE (%Module)	#!Generate module
header	
#FOR (%ModuleProcedure)	#!For all procs in module
#FIX (%Procedure,%ModuleProcedure)	#!Fix current procedure
#GENERATE (%Procedure)	#!Generate
procedure code	
#ENDFOR	#!EndFor all procs in
module	
#ENDFOR	#!EndFor all modules

#ABORT (прекращение генерации)

#ABORT

Оператор #ABORT немедленно прекращает генерацию исходного текста предыдущим оператором #GENERATE. Оператор #ABORT может размещаться в любой шаблонной секции.

Пример:

```
#IF (%ValidRangeKey=%Null)
  #ERROR(%Procedure & ' Range Error: The range field is not in the primary
key!')
  #ABORT
#ENDIF
```

См. также: #GENERATE

Операторы управления файлами

#CREATE (создание файла исходного текста)

#CREATE(*file*)

#CREATE

Создает дисковый файл для записи сгенерированного исходного текста.

file Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.

Оператор #CREATE создает дисковый файл, который будет получать исходный текст, сгенерированный оператором #GENERATE. Если file не существует, он создается. Если file уже существует, он открывается и очищается (усекается до нулевой длины). Если file уже открыт, при генерации текста формируется ошибка.

Файл file автоматически выбирается в качестве активного приемника выходного текста.

Пример:

```
#SET (%NewProgramFile, (%Application & '.$$$'))    #!Temp new program filename
#CREATE(%NewProgramFile)                          #!Create new program
file
#GENERATE (%Program)                               #!Generate main
program header
```

#OPEN (открыть файл исходного текста)

#OPEN (*file*) [, READ]

#OPEN	Открывает дисковый файл для записи сгенерированного исходного текста.
<i>file</i>	Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.
READ	Открывает файл в режиме “только чтение”. Файл не может быть открытым ранее для вывода.

Оператор #OPEN открывает дисковый файл, который будет получать исходный текст, сгенерированный оператором #GENERATE. Если file не существует, он создается. Если file уже существует, он открывается в режиме “дописывания текста”. Если file уже открыт, при генерации текста формируется ошибка. Файл file автоматически выбирается в качестве активного приемника выходного текста.

Если присутствует атрибут READ, файл открывается в режиме “только чтение”, так что оператор #READ может только читать его как текстовый ASCII файл. Только один файл может быть открыт для вывода в любой момент времени.

Пример:

```
#SET (%ProgramFile, (%Application & '.$$$')) #!Temp program filename
#OPEN(%ProgramFile)                        #!Open existing program file
#GENERATE (%Program)                       #!Generate main program
header
#CLOSE(%ProgramFile)                      #!Close output file
```

```

#OPEN (%ProgramFile),READ          #!Open it in read-only mode
#DECLARE      (%ASCIIFileRecord)
#LOOP
    #READ(%ASCIIFileRecord)          #! Parse the line and do something with
it
    #IF (%ASCIIFileRecord = %EOF)
        #BREAK
    #ENDIF
#ENDLOOP
#CLOSE(%ProgramFile),READ
См. также: #READ, #CLOSE

```

#CLOSE (закрывать файл исходного текста)

#CLOSE([*file*]) [, READ]

#CLOSE Закрывает открытый файл с сгенерированным исходным текстом.
file Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS. Если опущено, закрывается текущий дисковый файл, получающий генерируемый исходный текст.

READ Закрывает входной файл, открытый в режиме “только чтение”.

Оператор #CLOSE закрывает открытый дисковый файл, получающий генерируемый исходный текст. Если *file* опущено, закрывается текущий дисковый файл, получающий генерируемый исходный текст. Если *file* не существует или уже закрыт, при генерации текста формируется ошибка.

Пример:

```

#SET (%NewProgramFile, (%Application & '.$$$'))    #!Temp new program filename
#CREATE(%NewProgramFile)                          #!Create new program file
#GENERATE (%Program)                               #!Generate main program
header
#CLOSE(%NewProgramFile)                            #!Create new program file

#OPEN (%ProgramFile),READ                          #!Open it in read-only mode
#DECLARE (%ASCIIFileRecord)
#LOOP
    #READ(%ASCIIFileRecord)          #! Parse the line and do something with
it
    #IF (%ASCIIFileRecord = %EOF)

```

```
#BREAK
#ENDIF
#ENDLOOP
#CLOSE(%ProgramFile),READ          #!Close the read-only file
См. также: #OPEN, #READ
```

#READ(чтение одной строки из файла исходного текста)

#READ(*symbol*)

#READ Считывает следующую запись из файла, открытого в режиме “только чтение”.

symbol Символ, получающий текст из файла.

Оператор #READ считывает следующую запись (до следующего встреченного CR/LF) из файла, открытого в режиме “только чтение”. Символ *symbol* получает текст из файла. Если была прочитана последняя запись, *symbol* будет содержать значение, эквивалентное встроенному символу %EOF.

Пример:

```
#OPEN(%ProgramFile),READ          #!Open it in read-only mode
#DECLARE      (%ASCIIFileRecord)
#LOOP
  #READ(%ASCIIFileRecord)          #!Parse the line and do something with
it
  #IF (%ASCIIFileRecord = %EOF)
    #BREAK
  #ENDIF
#ENDLOOP
#CLOSE(%ProgramFile),READ          #!Close the read-only file
См. также: #OPEN, #CLOSE
```

#REDIRECT(изменить файл исходного текста)

#REDIRECT([*file*])

#REDIRECT Изменяет текущий файл, получающий сгенерированный исходный текст.

file Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS, который был ранее открыт операторами #OPEN или #CREATE. Здесь может содержаться полное имя файла DOS. Если опущено, генерация исходного текста возвращается к тому файлу, который предыдущим получал сгенерированный исходный текст.

Оператор #REDIRECT заменяет файл, получающий генерируемый текст. Весь выход генерации текста направляется в указанный file, до тех пор, пока не будет выполнен оператор #OPEN или #REDIRECT. Если файл не был ранее открыт (или создан) или уже закрыт, при генерации текста формируется ошибка.

Целевые файлы для генерируемого текста хранятся в списке типа “стек” (LIFO - Last In, First Out). Когда выполняется оператор #REDIRECT без параметров, генерация исходного текста возвращается к предыдущему файлу.

Пример:

```
#SET (%NewProgramFile, (%Application & '.CLW'))    #!Temp new program filename
#CREATE(%NewProgramFile)                          #!Create new program
file
#FOR (%Module)
    #CREATE(%Module & '.CLW')                      #!Make module files
#ENDFOR
#REDIRECT(%NewProgramFile)                        #!Redirect output to program file
#GENERATE (%Program)                              #!Generate main program header
#CLOSE(%NewProgramFile)                          #!Create new program file
#FOR (%Module)
    #REDIRECT(%Module & '.CLW')                    #!Redirect output to module file
    #GENERATE (%Module)                            #!Generate module header
    #FOR (%ModuleProcedure)                        #!For all procs in module
        #FIX (%Procedure,%ModuleProcedure)        #!Fix current procedure
        #GENERATE (%Procedure)                    #!Generate procedure code
    #ENDFOR                                        #!EndFor all procs in module
#ENDFOR
#!The following code demonstrates the LIFO files list:
#REDIRECT('F1.CLW')    #!List contains:  F1
#REDIRECT('F2.CLW')    #!List contains:  F1, F2
#REDIRECT('F3.CLW')    #!List contains:  F1, F2, F3
#REDIRECT( )           #!List contains:  F1, F2
#REDIRECT( )           #!List contains:  F1
```

#APPEND (добавить в конец файла исходного текста)

#APPEND(*file*)

#APPEND

Добавляет содержимое файла *file* в конец текущего выходного файла исходного текста.

file Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.

Оператор **#APPEND** добавляет целиком содержимое файла *file* в конец текущего выходного файла исходного текста. Содержимое *file* НЕ интерпретируется для целей генерации исходного текста. Следовательно, *file* не должен содержать какой-либо текст на языке шаблонов.

Если файл не существует, оператор **#APPEND** игнорируется и генерация текста продолжается.

Пример:

```
#FOR (%Module)
  #SET (%TempModuleFile,(%Module & '.$$$'))      #!Set temp module file
  #CREATE(%TempModuleFile)                        #!Create temp module file
  #FOR (%ModuleProcedure)                          #!For all procs in module
    #FIX (%Procedure,%ModuleProcedure)            #!Fix current procedure
    #GENERATE (%Procedure)                         #!Generate procedure code
  #ENDFOR                                           #!EndFor all procs in module
  #SET (%ModuleFile,(%Module & '.CLW'))            #!Set to current module file
  #CREATE(%ModuleFile)                             #!Create module file
  #GENERATE (%Module)                              #!Generate module header
  #APPEND(%TempModuleFile)                         #!Add generated procedures
#ENDFOR
```

#REMOVE (удаление файла исходного текста)

#REMOVE(*file*)

#REMOVE Удаляет файл исходного текста.
file Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.

Оператор **#REMOVE** уничтожает указанный файл *file* исходного текста. Если *file* не существует, оператор **#REMOVE** игнорируется, и генерация текста продолжается.

Пример:

```
#FOR (%Module)
  #SET (%TempModuleFile,(%Module & '.$$$'))      #!Set temp module file
  #CREATE(%TempModuleFile)                        #!Create temp module file
  #FOR (%ModuleProcedure)                          #!For all procs in module
    #FIX (%Procedure,%ModuleProcedure)            #!Fix current procedure
    #GENERATE (%Procedure)                         #!Generate procedure code
  #ENDFOR                                           #!EndFor all procs in module
  #SET (%ModuleFile,(%Module & '.CLW'))            #!Set to current module file
  #CREATE(%ModuleFile)                             #!Create module file
```

#GENERATE (%Module)	#!Generate module header
#APPEND(%TempModuleFile)	#!Add generated procedures
#REMOVE(%TempModuleFile)	#!Delete the temporary file
#ENDFOR	

#REPLACE (условная замена файла исходного текста)

#REPLACE(*oldfile*, *newfile*)

#REPLACE	Выполняет “разумную” замену файла.
<i>oldfile</i>	Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.
<i>newfile</i>	Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя файла DOS.

Оператор #REPLACE выполняет двоичное сравнение содержимого *oldfile* и *newfile*. Если содержимое *oldfile* отличается от содержимого *newfile* (или *oldfile* не существует), то *oldfile* уничтожается, а *newfile* переименовывается в *oldfile*. Если два файла идентичны, то ничего не делается. Если *newfile* не существует, #REPLACE игнорируется и генерация исходного текста продолжается.

Пример:

#FOR (%Module)	
#SET (%TempModuleFile,(%Module & ‘.\$\$\$’))	#!Set temp module file
#CREATE(%TempModuleFile)	#!Create temp module file
#GENERATE (%Module)	#!Generate module header
#FOR (%ModuleProcedure)	#!For all procs in module
#FIX (%Procedure,%ModuleProcedure)	#!Fix current procedure
#GENERATE (%Procedure)	#!Generate procedure code
#ENDFOR	#!EndFor all procs in module
#SET (%ModuleFile,(%Module & ‘.CLW’))	#!Set to existing module file
#REPLACE(%ModuleFile,%TempModuleFile)	#!Replace old with new (if hanged)
#ENDFOR	

#PRINT (печать файла исходного текста)

#PRINT(*file*, *title*)

#PRINT	Печатает <i>file</i> на текущий принтер Windows.
<i>file</i>	Строковая константа, шаблонный символ или выражение, содержащее спецификацию файла DOS. Здесь может содержаться полное имя

title файла DOS.
Строковая константа, шаблонный символ или выражение, содержащее заголовок file.

Оператор #PRINT печатает текущее содержимое file на стандартный принтер Windows.

Пример:

```
#FOR (%Module)
    #SET (%ModuleFile,(%Module & '.CLW'))    #!Set to existing module file
    #PRINT(%ModuleFile,"Printout ' & %ModuleFile)
#ENDFOR
```

Операторы условной генерации

#SUSPEND (начать условный текст)

#SUSPEND

Оператор #SUSPEND отмечает начало секции исходного текста, которая будет генерироваться только, если встретится оператор #RELEASE. Это позволяет очистить и легко удалить из генерируемого текста ненужный фрагмент. Окончание секции должно быть обозначено соответствующим оператором #RESUME.

Эти секции #SUSPEND могут быть вложены друг в друга на столько уровней, сколько необходимо. Встречающийся во внутренней вложенной секции оператор #RELEASE подтверждает генерацию текста и для всех внешних уровней, в которые он вложен.

Оператор #EMBED, содержащий текст для генерации, выполняет неявный #RELEASE. Любой вывод сгенерированного текста также выполняет неявный #RELEASE. Следовательно, явный оператор #RELEASE не всегда необходим. Оператор #? задает отдельную условно генерируемую строку, которая не выполняет неявный #RELEASE.

Пример:

```
ACCEPT
#SUSPEND    #!Begin suspended generation
    #?CASE SELECTED()
    #FOR (%ScreenField)
        #SUSPEND
    #?OF %ScreenField
        #EMBED (%ScreenSetups,'Control selected code'),%ScreenField
```

```

    #!Implied #RELEASE from the #EMBED of both nested sections
    #RESUME
END
#RESUME  #!End suspended generation
#SUSPEND #!Begin suspended generation
    #?CASE EVENT()
    #SUSPEND
    #?OF EVENT:AlertKey
        #SUSPEND
        #?CASE KEYCODE ( )
            #FOR %HotKey
                #RELEASE #!Explicit #RELEASE
            #?OF %HotKey
                #EMBED (%HotKeyProc,'Hot Key code'),%HotKey
            #ENDFOR
        #?END
    #RESUME
    #?END
#RESUME  #!End suspended generation
END
См. также:  #RELEASE,#RESUME,#?

```

#RELEASE (подтверждение условной генерации текста)

#RELEASE

Оператор #RELEASE разрешает генерацию исходного текста в секции #SUSPEND. Это позволяет очистить и легко удалить из генерируемого текста ненужный фрагмент. Текст в секции #SUSPEND генерируется только, если встречается оператор #RELEASE.

Секции #SUSPEND могут быть вложены друг в друга на столько уровней, сколько необходимо. Встречающийся во внутренней вложенной секции оператор #RELEASE подтверждает генерацию текста и для всех внешних уровней, в которые он вложен.

Оператор #EMBED, содержащий текст для генерации, выполняет неявный #RELEASE. Любой вывод сгенерированного текста также выполняет неявный #RELEASE. Следовательно, явный оператор #RELEASE не всегда необходим. Оператор #? задает отдельную условно генерируемую строку, которая не выполняет неявный #RELEASE.

Пример:

```

ACCEPT
#SUSPEND  #!Begin suspended generation

```

```
#?CASE SELECTED()
#FOR (%ScreenField)
  #SUSPEND
#?OF %ScreenField
  #EMBED (%ScreenSetups,'Control selected code'),%ScreenField
  #!Implied #RELEASE from the #EMBED of both nested sections
  #RESUME
#?END
#RESUME  #!End suspended generation
#SUSPEND  #!Begin suspended generation
  #?CASE EVENT()
  #SUSPEND
  #?OF EVENT:AlertKey
    #SUSPEND
    #?CASE KEYCODE ( )
      #FOR %HotKey
        #RELEASE #!Explicit #RELEASE
      #?OF %HotKey
        #EMBED (%HotKeyProc,'Hot Key code'),%HotKey
      #ENDFOR
    #?END
  #RESUME
#?END
#RESUME  #!End suspended generation
END
См. также:  #SUSPEND,#RESUME,#?
```

#RESUME (граница условной генерации текста)

#RESUME

Оператор #RESUME отмечает конец секции исходного текста, которая будет генерироваться только, если встретится оператор #RELEASE. Это позволяет очистить и легко удалить из генерируемого текста ненужный фрагмент. Начало секции должно быть обозначено соответствующим оператором #SUSPEND.

Эти секции #SUSPEND могут быть вложены друг в друга на столько уровней, сколько необходимо. Встречающийся во внутренней вложенной секции оператор #RELEASE подтверждает генерацию текста и для всех внешних уровней, в которые он вложен.

Оператор #EMBED, содержащий текст для генерации, выполняет неявный #RELEASE. Любой вывод сгенерированного текста также выполняет неявный #RELEASE. Следовательно, явный оператор #RELEASE не всегда необходим. Оператор #? задает отдельную

условно генерируемую строку, которая не выполняет неявный #RELEASE.

Когда #RESUME выполняется без подтверждения выводом в файл, все условные строки текста откатываются к соответствующему #SUSPEND.

Пример:

```

ACCEPT
#SUSPEND                                #!Begin suspended generation
  #?CASE SELECTED()
  #FOR (%ScreenField)
    #SUSPEND
  #?OF %ScreenField
    #EMBED (%ScreenSetups,'Control selected code'),%ScreenField
  #!Implied #RELEASE from the #EMBED of both nested sections
  #RESUME
  #?END
#RESUME                                #!End suspended generation
#SUSPEND                                #!Begin suspended generation
  #?CASE EVENT()
  #SUSPEND
  #?OF EVENT:AlertKey
    #SUSPEND
    #?CASE KEYCODE ( )
      #FOR %HotKey
        #RELEASE                        #!Explicit #RELEASE
      #?OF %HotKey
        #EMBED (%HotKeyProc,'Hot Key code'),%HotKey
      #ENDFOR
    #?END
    #RESUME
  #?END
#RESUME                                #!End suspended generation
END
См. также:  #SUSPEND,#RELEASE,#?

```

#? (условная строка условного текста)

#?statement

#?	Определяет отдельную строку исходного текста, которая генерируется только, если оператор #RELEASE подтверждает условную секцию исходного текста.
<i>statement</i>	Отдельная строка целевого языка. Она может содержать шаблонные символы.

Оператор `#?` определяет отдельную строку исходного текста, которая генерируется только, если оператор `#RELEASE` подтверждает условную секцию исходного текста. Это позволяет очистить и легко удалить из генерируемого текста ненужный фрагмент.

Оператор `#EMBED`, содержащий текст для генерации, выполняет неявный `#RELEASE`. Любой вывод сгенерированного текста также выполняет неявный `#RELEASE`. Следовательно, явный оператор `#RELEASE` не всегда необходим. Оператор `#?` задает отдельную условно генерируемую строку, которая не выполняет неявный `#RELEASE`.

Когда `#RESUME` выполняется без подтверждения выводом в файл, все условные строки текста откатываются к соответствующему `#SUSPEND`.

Пример:

```
ACCEPT                                #!Unconditional source line
#SUSPEND
  #?CASE SELECTED()                  #!Conditional source line
  #FOR (%ScreenField)
    #SUSPEND
    #?OF %ScreenField                #!Conditional source line
    #EMBED (%ScreenSetups,'Control selected code'),%ScreenField
    #RESUME
    #?END                            #!Conditional source line
#RESUME
#SUSPEND
  #?CASE EVENT()                     #!Conditional source line
  #SUSPEND
  #?OF EVENT:AlertKey                #!Conditional source line
  #SUSPEND
  #?CASE KEYCODE ()                 #!Conditional source line
    #FOR %HotKey
      #RELEASE
    #?OF %HotKey                     #!Conditional source line
    #EMBED (%HotKeyProc,'Hot Key code'),%HotKey
    #ENDFOR
    #?END                            #!Conditional source line
  #RESUME
  #?END                            #!Conditional source line
#RESUME
E N D                                #!Unconditional source line
```

См. также: `#SUSPEND`, `#RELEASE`, `#RESUME`

Прочие операторы

Прочие операторы

#! (комментарии в тексте шаблона)
#< (выровненные комментарии целевого языка)
#CLASS (определение класса формулы)
#COMMENT (определение колонки комментария)
#ERROR (вывод ошибок генерации исходного текста)
#HELP (определение файла справочника по шаблонам)
#INCLUDE (включение файла шаблона)
#IMPORT (импорт .APP из сценария)
#MESSAGE (вывод сообщения о генерации исходного текста)
#PROTOTYPE (прототип процедуры)
#PROJECT (добавление файла к проекту)

Встроенные шаблонные функции

EXTRACT (атрибут)
EXISTS (существование точки вставки)
FILEEXISTS (существование файла)
INLIST (существование элемента списка)
INSTANCE (текущий номер экземпляра символа)
ITEMS (количество экземпляров многозначного символа)
SEPARATOR (положение разделителя в строке атрибутов)

Прочие операторы

#! (комментарии в тексте шаблона)

#! comments

#! Начинает комментарий языка шаблонов.
comments Произвольный текст.

Комментарии в языке шаблонов начинаются с символов **#!**. Весь текст справа от символов **#!** до конца строки игнорируется процессором шаблонов. **#!**-комментарии не включаются в сгенерированный исходный текст.

Пример:

```
#! These are Template comments which  
#! will not end up in the generated code
```

#< (выровненные комментарии целевого языка)

#<comments

#<
comments Начинает выровненный комментарий целевого языка.
 Произвольный текст. Он должен начинаться с признака комментария
 целевого языка.

Комментарии целевого языка, которые включаются в сгенерированный текст, начинаются с символов **#<**. Комментарии генерируются начиная с колонки, указанной в операторе **#COMMENT**. Если позиция колонки занята, комментарии приписываются справа через один пробел к оператору сгенерированного текста. Любой стандартный комментарий целевого языка без предшествующих **#<** включается в сгенерированный текст в той же колонке, которую он занимает в шаблоне.

Пример:

```
#COMMENT (50) #! This Template file comment will not be in the generated  
              code  
#<! This is a Clarion comment which appears in the generated code in column 50  
  ! This Clarion comment appears in the generated code in column 2  
#<// This is a C++ comment which appears in the generated code beginning in  
column 50
```

// This C++ comment appears in the generated code in column 2
 См. также: #COMMENT

#CLASS (определение класса формулы)

#CLASS(*string*, *description*)

#CLASS	Определяет класс формулы.
<i>string</i>	Строковая константа, содержащая класс формулы.
<i>description</i>	Строковое выражение, содержащее описание класса формулы для вывода в списке доступных формул в Редакторе Формул.

Оператор #CLASS определяет класс формулы для использования редактором формул. Класс формулы позволяет шаблонам определять точную логическую позицию, в которой формулы появляются в сгенерированном тексте.

Пример:

```
#PROCEDURE (SomeProc,'An Example Template'),WINDOW
#CLASS('START','At beginning of procedure')
#CLASS('LOOP','In process loop')
#CLASS('END','At end of procedure')
%Procedure PROCEDURE
%ScreenStructure
CODE
#INSERT (%GenerateFormulas,'START') #!Generate START class formulas
OPEN(%Screen)
ACCEPT
    #INSERT (%GenerateFormulas,'LOOP') #!Generate LOOP class formulas
END
    #INSERT (%GenerateFormulas,'END') #!Generate END class formulas
```

#COMMENT (определение колонки комментария)

#COMMENT(*column*)

#COMMENT	Задаёт стандартный номер колонки для выравнивания комментариев.
<i>column</i>	Числовая константа в диапазоне от 1 до 255.

Оператор #COMMENT задаёт стандартный номер в колонки, в которой Генератор Приложений будет генерировать комментарии Clariion, начинающиеся с оператора #<!.

Пример:

```
#COMMENT (50)    #!Set comment column
IF Action = 1    #<!If adding a record
    SomeVariable = InitVariable
END
См. также:    #<
```

#ERROR (вывод ошибок генерации исходного текста)

#ERROR (*message*)

#ERROR Выводит ошибку генерации исходного текста.
message Строковая константа, определяемый символ или выражение, содержащее сообщение об ошибке для вывода в окне Source Generation.

Оператор #ERROR выводит сообщение *message* в окне Source Generation. Это может быть информация для пользователя. Это может также предупредить пользователя, что он сделал некоторые ошибки, которые привели к генерации процедурным шаблоном неправильного исходного текста, что может привести к ошибкам компилятора.

Когда оператор #ERROR встречается в исходном тексте во время генерации, выводится его сообщение. Пользователь может выбрать прекращение компиляции и сборки или продолжить компиляцию.

Пример:

```
#PROCEDURE (SampleProc,'This is a sample procedure')
#PROMPT ('Access Key',KEY),%SampleAccessKey
  #IF (%SampleAccessKey = %NULL)    #!IF the user did not enter a Key
    #SET (%ErrorSymbol,(%Procedure & ' Access Key blank')
    #ERROR (%ErrorSymbol)
    #ERROR ('This error is Fatal — DO NOT CONTINUE')
    #ABORT
#ENDIF
```

#EXPORT (вывести символ в текст)

#EXPORT(*symbol*)

EXPORT Создает .TXA текст из содержимого *symbol*.
symbol Экспортируемый шаблонный символ.

Оператор #EXPORT выводит в текущий выходной файл (см. #CREATE или #OPEN) текст описания в .TXA формате содержимого *symbol*. Этот .TXA файл может быть затем импортирован в другие Clarion приложения.

Пример:

```
#OPEN('MyExp.TXA')
#FOR(%Procedure)
    #EXPORT(%Procedure)
#ENDFOR
```

См. также: #CREATE, #OPEN, #IMPORT

#HELP (определение файла справочника по шаблонам)

#HELP(*helpfile*)

#HELP	Задаёт файл справочника по шаблонам.
<i>helpfile</i>	Строковая константа, содержащая имя файла справочника по шаблонам.

Оператор #HELP задаёт *helpfile*, который используется шаблоном. Будучи определённым, *helpfile* используется для обращения к разделам справки, указанным идентификатором справки во всех атрибутах HLP шаблона.

Пример:

```
#HELP('Template.HLP')
```

#INCLUDE (включение файла шаблона)

#INCLUDE (*filename*)

#INCLUDE	Добавляет файл шаблона к цепочке файлов шаблона.
<i>filename</i>	Строковая константа, содержащая имя включаемого шаблонного файла.

Оператор #INCLUDE добавляет шаблонный файл в цепочку шаблонных файлов. Продолжение шаблонного файла, содержащего оператор #INCLUDE будет обработано после обработки включаемого файла.

Пример:

```
#TEMPLATE (Clarion,'Clarion Standard Shipping Templates')
#INCLUDE ('Clarion1.TPX') #!Include a template file
#INCLUDE ('Clarion2.TPX') #!Include another template file
```

#IMPORT (импорт .APP из сценария)

```
#IMPORT( source ) [, | RENAME | ]
                        | REPLACE |
```

#IMPORT <i>source</i>	Создает .APP для Clarion for Windows из .TXA файла сценария. Имя .TXA файла сценария, из которого будет создаваться .APP файл.
RENAME	Переопределяет установку Procedure Name Clash и переименовывает все процедуры.
REPLACE	Переопределяет установку Procedure Name Clash и заменяет все процедуры.

Оператор #IMPORT добавляет определение процедуры и/или функции к Clarion for Windows .APP файлу из .TXA файла сценария. Это используется для импорта из других версий продуктов Clarion для разработки приложений.

Пример:

```
#UTILITY (SomeUtility,'Some Utility Template')
  #PROMPT ('File to import',@s64),%ImportFile
  #IMPORT(%ImportFile)
```

#MESSAGE (вывод сообщения о генерации исходного текста)

```
#MESSAGE( message, line )
```

#MESSAGE <i>message</i>	Выводит сообщение о генерации исходного текста. Строковая константа или определяемый символ, содержащий сообщение для вывода в диалоге Source Generation.
<i>line</i>	Целая константа или символ, содержащий номер строки, в которой будет выводиться message. Если вне диапазона от 1 до 3, message будет выводиться в заголовке окна.

Оператор #MESSAGE выводит message в диалоге Source Generation. Первый оператор #MESSAGE выводит заголовок окна. Последующие операторы #MESSAGE изменяют выводимый текст.

Пример:

```
#MESSAGE ('Generating ' & %Application,0) #!Display Title bar text
#MESSAGE ('Generating ' & %Procedure,2) #!Display Progress message on line 2
```

#PROTOTYPE (прототип процедуры)**#PROTOTYPE (*parameter list*)**

#PROTOTYPE	Присваивает список параметров <i>parameter list</i> вводимому полю <i>Prototype</i> .
<i>parameter list</i>	Строковая константа, содержащая список параметров прототипа процедуры (весь прототип процедуры за исключением имени процедуры) для структуры MAP приложения (см. обсуждение прототипов процедур и функций в описании языка).

Оператор **#PROTOTYPE** присваивает *parameter list* вводимому полю *Prototype* на окне *Procedure Properties* Генератора Приложений, что автоматически блокирует это поле (программист не может переопределить этот прототип). Это позволяет создать процедурный шаблон, который требует специального списка параметров, не требуя от программиста знания прототипа процедуры.

Оператор **#PROTOTYPE** может находиться только в секции **#PROCEDURE** и в одной секции **#PROCEDURE** может находиться только один оператор. Если в **#PROCEDURE** нет оператора **#PROTOTYPE**, программист может изменить его.

Пример:

```
#PROCEDURE (SomeProc,'Some Procedure Template')
%Procedure PROCEDURE (Parm1,Parm2,Parm3)
  #PROTOTYPE (('STRING,*LONG,<*SHORT>'))
    #!This procedure expects three parameters:
    #! a STRING passed by value
    #! a LONG passed by address
    #! a SHORT passed by address which may be omitted

#PROCEDURE (SomeFunc,'Some Template Function')
%Procedure FUNCTION(Parm1,Parm2,Parm3)
  #PROTOTYPE (('STRING,*LONG,<*SHORT>'),STRING')
    #!This function expects three parameters:
    #! a STRING passed by value
    #! a LONG passed by address
    #! a SHORT passed by address which may be omitted
    #!It returns a STRING
```

#PROJECT (добавление файла к проекту)

#PROJECT (*module*)

#PROJECT

Включает исходный текст или библиотеку в объектном коде или проектный файл в проектный файл (.PRJ) приложения.

module

Строковая константа, которая задает имя исходного (.CLW, если целевым языком является Clarion), объектного (.OBJ) или библиотечного (.LIB) файла, содержащего процедуры и/или функции, требуемые процедурным шаблоном. Здесь также может быть указано имя проектного (.PRJ) файла, вызываемого проектом приложения. Тип включаемого файла определяется по его расширению.

Оператор #PROJECT указывает на исходную или объектную библиотеку, которая должна быть в приложении для правильной работы процедур, созданных процедурным шаблоном.

Оператор #PROJECT обеспечивает прямой метод подключения информации модулей *module* к Генератору Приложений и проектной системе. Он оповещает Генератор Приложений о необходимости присутствия *module* для компиляции и/или сборки приложения. Следовательно, проектный файл приложения (созданный Генератором Приложений) автоматически включает *module* для создания, компиляции и/или сборки.

Если создаваемая в приложении процедура несколько раз ссылается на один и тот же экземпляр оператора #PROJECT, используется только первая ссылка. Это могло бы иметь место, когда многие процедурные шаблоны или несколько процедур приложения, созданных из одного и того же процедурного шаблона, требовали бы одного и того же *module*.

Оператор #PROJECT позволяет разработчику автоматизировать установку библиотек и шаблонов третьих лиц на компьютеры других разработчиков. Это обеспечивает правильную генерацию проекта приложения.

Оператор #PROJECT дает возможность создать в проектном файле (.PRJ) для больших разрабатываемых проектов иерархию проектов. Когда многочисленные библиотеки собираются в пакет, то это позволяет удостовериться, что выполняются все зависимости для всех библиотек, на которые есть ссылки в частном проекте.

Пример:

```
#AT(%CustomGlobalDeclarations)
    #PROJECT('Party3.LIB')
#ENDAT
```


Встроенные шаблонные функции

EXTRACT (атрибут)

EXTRACT(*string*, *attribute* [, *parameter*])

EXTRACT	Возвращает полную форму указанного атрибута <i>attribute</i> для свойства символа, заданного строкой <i>string</i> .
<i>string</i>	Символ, содержащий исследуемые свойства.
<i>attribute</i>	Строковая константа или символ, содержащий имя возвращаемого свойства.
<i>parameter</i>	Целочисленная константа или символ, содержащий номер параметра для свойства. Ноль (0) возвращает полный список параметров (без <i>attribute</i>). Если не указан, возвращается <i>attribute</i> со списком всех параметров.

Функция EXTRACT возвращает либо полную форму указанного атрибута из строки атрибутов символа, либо только указанный параметр. Это применяется, если нет встроенных символов для нужного вам атрибута.

Возвращаемый тип данных: **STRING**

Пример:

```
#SET (%MySymbol,EXTRACT(%ControlStatement,'DROPID') #!Return DROPID
attribute
#SET(%MySymbol,EXTRACT(%ControlStatement,'DROPID',0)
#!Return all DROPID parameters
```

См. также: REPLACE

EXISTS (существование точки вставки)

EXISTS(*symbol*)

EXISTS	Возвращает TRUE, если точка вставки исходного текста доступна для использования.
<i>symbol</i>	Символ identifier оператора #EMBED, задающего точку вставки.

Функция EXISTS возвращает истину ('1'), если указанная точка вставки исходного текста доступна для использования во время разработки. Если указанная точка вставки исходного текста не доступна для использования функция EXISTS возвращает ложь (''). Точка вставки исходного текста является доступной, если используется содержащая ее секция. Это означает, что все #EMBED во всех секциях #PROCEDURE всегда доступны. Точки вставки #EMBED в секциях #CONTROL, #CODE или #EXTENSION

доступны только, если секции используются.

Возвращаемый тип данных: LONG

Пример:

```
#IF (EXISTS(%CodeTemplateEmbed))
!Generate some source
#ENDIF
```

FILEEXISTS (существование файла)

FILEEXISTS(*file*)

FILEEXISTS Возвращает TRUE, если *file* доступен на диске.
file Выражение, содержащее имя файла DOS.

Функция FILEEXISTS возвращает истину ('1'), если файл *file* доступен на диске. Если файл *file* недоступен, FILEEXISTS возвращает ложь ('').

Возвращаемый тип данных: LONG

Пример:

```
#IF (FILEEXISTS(%SomeFile))
#OPEN (%SomeFile)
#READ(%SomeFile)
!some source
#ENDIF
```

INLIST (существование элемента списка)

INLIST(*item*, *symbol*)

INLIST Возвращает номер экземпляра *item* в *symbol*.
item Строковая константа или символ, содержащий имя элемента.
symbol Многочисленный символ, который может содержать элемент.

Функция INLIST возвращает номер экземпляра *item* в *symbol*. Если *item* не содержится в *symbol*, INLIST возвращает (0).

Возвращаемый тип данных: LONG

Пример:

```
#IF (INLIST('?MyControl',%Control))
!Generate some source
#ENDIF
```

INSTANCE (текущий номер экземпляра символа)**INSTANCE (*symbol*)**

INSTANCE Возвращает текущий номер экземпляра, на котором зафиксирован *symbol*.
symbol Многозначный символ.

Функция INSTANCE возвращает текущий номер экземпляра, на котором зафиксирован *symbol*. Если для символа не выполнялись операторы #FIX или #FOR, INSTANCE возвращает (0).

Возвращаемый тип данных: LONG

Пример:

```
#DELETE (%Control,INSTANCE(%Control)) #!Delete current instance
```

ITEMS (количество экземпляров многозначного символа)**ITEMS(*symbol*)**

ITEMS Возвращает количество экземпляров, содержащихся в многозначном символе *symbol*.
symbol Многозначный символ.

Функция ITEMS возвращает количество экземпляров, содержащихся в многозначном символе *symbol*.

Возвращаемый тип данных: LONG

Пример:

```
#DELETE (%Control,ITEMS(%Control)) #!Delete last instance
```

LINKNAME (возвращает преобразованное имя процедуры)**LINKNAME(*prototype*)**

LINKNAME Возвращает преобразованное имя процедуры для редактора связей.
prototype Строковая константа или символ, содержащий прототип процедуры в том виде, как он представлен в MAP структуре.

Функция LINKNAME возвращает преобразованное имя процедуры для редактора связей.

Возвращаемый тип данных: STRING

Пример:

```
%(LINKNAME(%Procedure & %Prototype)) @%ExpLineNumber
```

QUOTE (замена в строке специальных символов)

QUOTE(*symbol*)

QUOTE Расширяет строковые данные символа, дублируя одиночные кавычки (') и непарные левые угловые (<) и фигурные ({) скобки для предотвращения ошибок компилятора.

symbol Символ, содержащий данные для грамматического разбора.

Функция QUOTE возвращает строку, содержащую строковые данные символа с продублированными одиночными кавычками (') и непарными левыми угловыми (<) и фигурными ({) скобками для предотвращения ошибок компилятора. Это дает возможность пользователю вводить строковые константы, содержащие апострофы, и выражения фильтра, содержащие знаки “меньше чем” (<), не требуя повторного ввода каждого из них.

Возвращаемый тип данных: STRING

Пример:

```
#PROMPT('Filter Expression',@S255),%FilterExpression
```

```
#SET(%ValueConstruct,QUOTE(%FilterExpression))
```

```
#!Expand single quotes and angle brackets
```

См. также:%'

REPLACE (замена атрибута)

REPLACE(*string, attribute, new value* [, *parameter*])

REPLACE Ищет полную форму указанного атрибута attribute в строке свойств string и заменяет его на new value.

string Символ, содержащий строку для грамматического разбора.

attribute Строковая константа или символ, содержащий имя искомого атрибута.

new value Строковая константа или символ, содержащий новое значение атрибута attribute.

parameter Целочисленная константа или символ, содержащий номер параметра для свойства. Ноль (0) возвращает полный список параметров (без attribute). Если не указан, возвращается attribute со списком всех параметров.

Функция REPLACE заменяет в строке символа string либо целиком указанный атрибут attribute, либо только указанный параметр parameter, на новое значение new value.

Возвращаемый тип данных: STRING.

Пример:

```
#SET(%ValueConstruct,REPLACE(%ValueConstruct,'MSG',''))
```

```
#!Remove MSG attribute
```

См. также:EXTRACT

SEPARATOR (положение разделителя в строке атрибутов)**SEPARATOR(*string*, *start*)**

SEPARATOR	Возвращает положение следующей запятой в строке атрибутов <i>string</i> .
<i>string</i>	Строковая константа или символ, содержащий разделенный запятыми список атрибутов.
<i>start</i>	Целочисленная константа или символ, содержащий начальную позицию для поиска следующей запятой.

Функция SEPARATOR возвращает положение следующей запятой в атрибуте *string* после позиции *start*. Эта функция корректно обрабатывает вложенные кавычки в *string*, так что запятые в строковых константах не приводят к возврату неправильной позиции.

Возвращаемый тип данных: LONG

Пример:

```
#SET (%MySymbol,SEPARATOR(%ControlStatement,1)
      #!Return first comma position
```


Символы шаблонов

Обзор символов

Символы расширения
Обзор иерархии символов

Встроенные символы

Символы, зависящие от %Application
Символы, зависящие от %File
Символы, зависящие от %ViewFiles
Символы, зависящие от %Field
Символы, зависящие от %Key
Символы, зависящие от %Relation
Символы, зависящие от %Module
Символы, зависящие от %Procedure
Символы оконных диалоговых элементов
Символы элементов данных документа
Символы формул
Символы файловой схемы
Символы файлового драйвера
Прочие символы

Обзор символов

В языке шаблонов Clarion for Windows используются символы, которые ведут себя как переменные в языках программирования - они содержат данные, которые могут использоваться как есть или могут входить в выражения. Это могут быть символы из встроенного набора или они могут быть определены автором шаблона. Оба вида могут быть как однозначными, так и многозначными.

Доступные разработчику шаблона встроенные символы содержат данные из Словаря Данных и Приложения в соответствии с тем, как программист разработал приложение. Определяемые шаблоном символы содержат данные, введенные программистом в диалогах свойств в Генераторе Приложений или могут быть внутреннего использования.

Все шаблонные символы при генерации раскрываются, чтобы разместить содержащиеся в них значения в генерируемом исходном тексте (если встречаются в тексте шаблона, который генерирует исходный текст).

Символы расширения

Есть несколько символов специального вида, раскрытие которых позволяет форматировать и вставлять специальные знаки в генерируемый текст. Для получения нужного результата они могут комбинироваться друг с другом.

%%	Раскрывается в одиночный знак процента (%). Это позволяет Генератору Приложений генерировать операцию деления по модулю, не конфликтуя с символами.
%#	Раскрывается в одиночный знак решетки (#). Это позволяет Генератору Приложений генерировать неявные LONG переменные, не конфликтуя с символами.
%@picture@symbol	Форматирует symbol по указанному шаблону в процессе генерации текста. Например, %@D1@MyDate раскрывает символ %MyDate, форматируя его по шаблону @D1.
%[number]symbol	Раскрывает symbol так, чтобы он занимал как минимум number позиций. Это позволяет обеспечить надлежащее выравнивание комментариев и типов данных при генерации текста.
%	Раскрывает очередной генерируемый текст в той же строке, что и предыдущий. Это символ продолжения строки шаблона.
%'symbol	Раскрывает symbol как текстовую константу в одиночных кавычках.
%(expression)	Раскрывает expression в генерируемый текст.

Пример:

Пример:

```
%(ALL(' ', %indent))%[20]Field %@d3@Date
    #!Generate an indent, expand %Field to occupy at least 20 spaces,
    #! then generate the date in mmm dd, yyyy format
```

```
%[30]Null    #!Generate 30 spaces
```

```
#! %MySymbol contains: Gavin's Holiday
```

```
StringVar = '%MySymbol'    #!Expands as a valid Clarion string constant
                        #! to 'Gavin's Holiday'
```

Обзор иерархии символов

Встроенные символы в целом образуют иерархические взаимозависимости. Эта иерархия ведет начало от %Application, от которого зависят все остальные встроенные символы. Следующая диаграмма не показывает все зависимые символы, но графически показывает иерархию символов. Большинство из них многозначны.

```
%Application
  %DictionaryFile
    %File
      %Field
      %Key
      %Relation
    %Program
    %GlobalData
    %Module
      %ModuleProcedure
      %MapItem
      %ModuleData
    %Procedure
      %Report
        %ReportControl
        %ReportControlField
      %Window
        %WindowEvent
        %Control
          %ControlEvent
      %ProcedureCalled
      %LocalData
```

%ActiveTemplate
%ActiveTemplateInstance
%Formula
%FormulaExpression

Эти символы (и все не показанные здесь, но зависящие от них) содержат все сведения о приложении, которые находятся в файлах словаря данных (.DCT) и приложения (.APP). Они позволяют написать шаблон для генерации любого требуемого текста.

Встроенные символы

Символы, зависящие от %Application

%Application	Спецификация файла .APP. Иерархия встроенных символов начинается с %Application.
%ApplicationDebug	Содержит 1, если включен отладочный режим.
%ApplicationLocal	
Library	Содержит 1, если в приложение собирается вместе с Clarion библиотекой периода исполнения.
%Target32	Содержит 1, если компилируется 32-разрядное приложение.
%DictionaryChanged	Содержит 1, если файл .DCT изменялся после последней генерации.
%RegistryChanged	Содержит 1, если файл REGISTRY.TRF изменялся после последней генерации.
%ProgramDate Created	Дата создания программы (стандартная дата Clarion).
%ProgramDate Changed	Дата последнего изменения программы (стандартная дата Clarion).
%ProgramTime Created	Время создания программы (стандартное время Clarion).
%ProgramTime Changed	Время последнего изменения программы (стандартное время Clarion).

%FirstProcedure	Метка первой процедуры приложения.
%HelpFile	Спецификация справочного файла приложения.
%ProgramExtension	Содержит EXE, DLL или LIB.
%DictionaryFile	Спецификация файла .DCT приложения.
%File	Содержит все объявления файлов в файле .DCT. Многозначный. Зависит от %DictionaryFile.
%Program	Спецификация файла PROGRAM (без расширения).
%GlobalData	Метки всех глобальных переменных, объявления которых сделаны в диалоге под кнопкой Global Data в диалоге Global Settings. Многозначный.
%GlobalDataStatement	Оператор объявления переменной (тип данного и все атрибуты). Зависит от %GlobalData.
%Module	Имена всех модулей исходного текста, кроме модуля PROGRAM. Многозначный.
%QuickProcedure	Тип процедуры, которую создает #UTILITY с атрибутом WIZARD.
%Procedure	Имена всех процедур и функций приложения. Многозначный.

Символы, зависящие от %File

%File	Содержит все объявления файлов в файле .DCT. Многозначный. Зависит от %DictionaryFile.
%FilePrefix	Содержимое атрибута PRE (префикс файла).
%FileDescription	Краткое описание файла.
%FileType	Содержит FILE, VIEW или ALIAS.
%FileDriver	Содержимое первого параметра атрибута DRIVER.

%FileDriverParameter	Содержимое второго параметра атрибута DRIVER.
%FileName	Содержимое атрибутов NAME операторов FILE.
%FileOwner	Содержимое атрибута OWNER.
%FileCreate	Содержит 1, если файл имеет атрибут CREATE.
%FileReclaim	Содержит 1, если файл имеет атрибут RECLAIM.
%FileEncrypt	Содержит 1, если файл имеет атрибут ENCRYPT.
%FileBindable	Содержит 1, если файл имеет атрибут BINDABLE.
%FileLongDesc	Полное описание файла.
%FileStruct	Оператор FILE (метка и все атрибуты).
%FileStructEnd	Ключевое слово END.
%FileStructRec	Оператор RECORD (включая метку и все атрибуты).
%FileStructRecEnd	Ключевое слово END.
%FileStatement	Содержит (только) атрибуты оператора FILE.
%FileThreaded	Содержит 1, если файл имеет атрибут THREAD.
%FileExternal	Содержит 1, если файл имеет атрибут EXTERNAL.
%FileExternalModule	Содержимое параметра файлового атрибута EXTERNAL.
%FilePrimaryKey	Метка первичного ключа файла.
%FileQuickOptions	Строка, содержащая разделенный запятыми список выбранных пользователем на страничке Options настроек файла.
%FileUserOptions	Строка, содержащая ввод пользователя в текстовое поле User Options на страничке Options настроек файла.
%ViewFilter	Содержимое атрибута FILTER.
%ViewStruct	Оператор VIEW (включая метку и все атрибуты).

<code>%ViewStructEnd</code>	Ключевое слово END.
<code>%ViewStatement</code>	Содержит (только) атрибуты оператора VIEW.
<code>%ViewPrimary</code>	Метка главного файла VIEW.
<code>%ViewPrimaryFields</code>	Метки всех полей главного файла VIEW. Многозначный.
<code>%ViewPrimaryField</code>	Зависит от <code>%ViewPrimaryFields</code> . Содержит метку поля главного файла VIEW.
<code>%ViewFiles</code>	Метки всех файлов VIEW. Многозначный.
<code>%AliasFile</code>	Метка файла-синонима ALIAS.
<code>%Field</code>	Метки всех полей файла (включая MEMO поля). Многозначный.
<code>%Key</code>	Метки всех ключей и индексов файла. Многозначный.
<code>%Relation</code>	Метки всех файлов, имеющих связь с текущим файлом. Многозначный.

Символы, зависящие от `%ViewFiles`

<code>%ViewFiles</code>	Метки всех файлов VIEW. Многозначный. Зависит от <code>%File</code> .
<code>%ViewFileStruct</code>	Оператор JOIN для присоединенного файла VIEW.
<code>%ViewFileStructEnd</code>	Ключевое слово END.
<code>%ViewFile</code>	Содержит метку файла.
<code>%ViewJoinedTo</code>	Метка файла к которому присоединен (JOIN) файл.
<code>%ViewFileFields</code>	Метки всех полей в файлах, использованных во VIEW. Многозначный.
<code>%ViewFileField</code>	Содержит метку поля файла, , использованного во VIEW.

Символы, зависящие от %Field

%Field	Метки всех полей файла (включая MEMO поля). Многозначный. Зависит от %File.
%FieldDescription	Краткое описание поля.
%FieldLongDesc	Полное описание поля.
%FieldFile	Метка файла, содержащего поле.
%FieldID	Метка поля без префикса.
%FieldDisplayPicture	Стандартный выводной шаблон.
%FieldRecordPicture	Определение шаблона хранения переменной STRING.
%FieldDimension1	Максимальное значение первой размерности массива.
%FieldDimension2	Максимальное значение второй размерности массива.
%FieldDimension3	Максимальное значение третьей размерности массива.
%FieldDimension4	Максимальное значение четвертой размерности массива.
%FieldHelpID	Содержимое атрибута HLP.
%FieldName	Содержимое атрибута NAME переменной.
%FieldRangeLow	Нижняя граница возможных значений переменной.
%FieldRangeHigh	Верхняя граница возможных значений переменной.
%FieldType	Тип данных переменной.
%FieldPlaces	Количество десятичных знаков в переменной.
%FieldMemoSize	Максимальный размер MEMO.
%FieldMemoImage	Содержит 1, если MEMO имеет атрибут BINARY.

%FieldInitial	Начальное значение переменной.
%FieldLookup	Файл для проверки значений переменной.
%FieldStruct	Оператор объявления переменной (метка, тип данных и все атрибуты).
%FieldStatement	Оператор объявления переменной (тип данных и все атрибуты).
%FieldHeader	Стандартный заголовок столбца в документах.
%FieldPicture	Стандартный формат вывода.
%FieldJustType	Содержит L, R, C или D для выравнивания переменной.
%FieldJustIndent	Величина смещения при выравнивании.
%FieldFormatWidth	Стандартная ширина диалогового элемента ENTRY для поля.
%FieldScreenControl	Диалоговый элемент для поля, как он задан в Словаре Данных.
%FieldScreenControlWidth	Либо стандартная, определяемая библиотекой периода исполнения, либо явно заданная в Словаре Данных ширина диалогового элемента для поля.
%FieldScreenControlHeight	Либо стандартная, определяемая библиотекой периода исполнения, либо явно заданная в Словаре Данных высота диалогового элемента для поля.
%FieldReportControl	Элемент документа для поля, как он задан в Словаре Данных.
%FieldReportControlWidth	Либо стандартная, определяемая библиотекой периода исполнения, либо явно заданная в Словаре Данных ширина элемента в документе для поля.
%FieldReportControlHeight	Либо стандартная, определяемая библиотекой периода исполнения, либо явно заданная в Словаре Данных высота элемента в документе для поля.
%FieldChoices	Варианты, которые пользователь ввел в поле Must Be In List. Многозначный.
%FieldQuickOptions	Строка, содержащая разделенный запятыми список выбранных пользователем на страничке Options настроек поля файла.
%FieldUserOptions	Строка, содержащая ввод пользователя в текстовое поле User

Options на страничке Options настроек поля файла.

Символы, зависящие от %Key

%Key	Метки всех ключей и индексов файла. Многозначный.
%KeyDescription	Краткое описание ключа.
%KeyLongDesc	Полное описание ключа.
%KeyFile	Метка файла к которому относится ключ.
%KeyID	Метка ключа (без префикса).
%KeyIndex	Содержит KEY, INDEX или DYNAMIC.
%KeyName	Содержимое атрибута NAME ключа.
%KeyAuto	Содержит метку поля с автоприращением.
%KeyDuplicate	Содержит 1, если ключ имеет атрибут DUP.
%KeyExcludeNulls	Содержит 1, если ключ имеет атрибут OPT.
%KeyNoCase	Содержит 1, если ключ имеет атрибут NOCASE.
%KeyPrimary	Содержит 1, если ключ описан в словаре как Primary.
%KeyStruct	Оператор объявления ключа (метка и все атрибуты).
%KeyStatement	Атрибуты ключа (только).
%KeyField	Метки всех входящих в ключ переменных. Многозначный.
%KeyFieldSequence	Содержит ASCENDING или DESCENDING. Зависит от %Keyfield.
%KeyQuickOptions	Строка, содержащая разделенный запятыми список выбранных пользователем на страничке Options настроек ключа файла.
%KeyUserOptions	Строка, содержащая ввод пользователя в текстовое поле User Options на страничке Options настроек ключа файла.

Символы, зависящие от %Relation

%Relation	Метки всех связанных с файлом файлов. Многозначный.
%RelationPrefix	Символы шаблоновПрефикс связанного файла.
%RelationAlias	Содержит имя файла, синоним к которому указан в %Relation.
%FileRelationType	Содержит 1:MANY или MANY:1.
%RelationKey	Метка ключа связанного файла.
%FileKey	Метка связывающего ключа файла.
%RelationConstraint Delete	Может содержать: RESTRICT, CASCADE или CLEAR.
%RelationConstraint Update	Может содержать: RESTRICT, CASCADE или CLEAR.
%RelationKeyField	Метки всех сопоставляемых переменных в ключе связанного файла. Многозначный.
%RelationKeyField Link	Метка сопоставляемой переменной в ключе файла. Зависит от %RelationKeyField.
%FileKeyField	Метки всех сопоставляемых переменных в ключе файла. Многозначный.
%FileKeyFieldLink	Метка всех сопоставляемых переменных в ключе связанного файла. Зависит от %FileKeyField.
%Relation QuickOptions	Строка, содержащая разделенный запятыми список выбранных пользователем на страничке Options настроек отношения.
%Relation UserOptions	Строка, содержащая ввод пользователя в текстовое поле User Options на страничке Options настроек отношения.

Символы, зависящие от %Module

%Module	Имена всех модулей исходного текста, кроме модуля PROGRAM. Многозначный.
%Module Description	Краткое описание модуля.
%ModuleLanguage	Содержит целевой язык модуля.
%ModuleTemplate	Имя используемого при генерации модуля шаблона Module.
%ModuleChanged	Содержит 1 если в модуле были изменения после последней генерации.
%ModuleExternal	Содержит 1, если это внешний модуль (не генерируется в Clarion for Windows).
%ModuleReadOnly	Содержит 1, если модуль имеет атрибут ReadOnly (только для чтения).
%ModuleExtension	Расширение имени файла модуля.
%ModuleBase	Имя модуля (без расширения).
%ModuleInclude	Файл, содержащий прототипы программ модуля, для включения оператором INCLUDE в структуру MAP.
%ModuleProcedure	Имена процедур и функций модуля. Многозначный.
%ModuleData	Метки переменных модуля, объявления которых сделаны в диалоге под кнопкой Data в диалоге Module Properties.
%ModuleData Statement	Оператор объявления переменной (тип данного и все атрибуты).

Символы, зависящие от %Procedure

%Procedure	Имена всех процедур и функций приложения. Многозначный.
%ProcedureType	Содержит PROCEDURE или FUNCTION.

%ProcedureReturnType	Возвращаемый тип данных, если подпрограмма является FUNCTION.
%ProcedureDateCreated	Дата создания процедуры (стандартная Clarion дата).
%ProcedureDateChanged	Дата последнего изменения процедуры (стандартная Clarion дата).
%ProcedureTimeCreated	Время создания процедуры (стандартное время Clarion).
%ProcedureTimeChanged	Время последнего изменения процедуры (стандартное время Clarion).
%ProcedureReadOnly	Содержит 1, если процедура имеет атрибут ReadOnly (только для чтения).
%Prototype	Прототип процедуры для MAP структуры.
%ProcedureTemplate	Имя шаблона Procedure, используемого для генерации процедуры.
%ProcedureDescription	Краткое описание процедуры.
%ProcedureExported	Содержит 1, если процедура входит в DLL, и может быть вызвана извне этого DLL.
%ProcedureLongDescription	Полное описание процедуры.
%ProcedureLanguage	Целевой язык, который генерирует процедурный шаблон.
%ProcedureCalled	Имена всех процедур, входящих в список процедур под кнопкой Procedures в диалоге Procedure Properties. Многозначный.
%LocalData	Метки всех переменных, объявления которых сделаны в диалоге под кнопкой Data в диалоге Procedure Properties. Многозначный.
%LocalDataStatement	Оператор объявления переменной (тип данного и все атрибуты). Зависит от %LocalData.
%ActiveTemplate	Имена всех использованных в процедуре диалоговых шаблонов. Многозначный.

%ActiveTemplateInstance	Номера экземпляров всех использованных в процедуре диалоговых шаблонов. Мнозначный. Зависит от %ActiveTemplate.
%ActiveTemplateParentInstance	Номер экземпляра диалогового шаблона с которому присоединен диалоговый шаблон. Это номер диалогового шаблона к которому непосредственно “прикреплен” текущий шаблон. Зависит от %ActiveTemplateInstance.
%ActiveTemplatePrimaryInstance	Номера экземпляра первоначального диалогового шаблона от которого зависит диалоговый шаблон. Это первый диалоговый шаблон в цепочке связанных диалоговых шаблонов. Зависит от %ActiveTemplateInstance.

Символы оконных диалоговых элементов

%Window	Метка окна процедуры. Зависит от %Procedure.
%WindowStatement	Оператор объявления структуры WINDOW или APPLICATION (со всеми атрибутами). Зависит от %Window.
%MenuBarStatement	Оператор объявления структуры MENUBAR (со всеми атрибутами). Зависит от %Window.
%ToolbarStatement	Оператор объявления структуры TOOLBAR (со всеми атрибутами). Зависит от %Window.
%WindowEvent	Все независящие от поля события, как перечислены в файле EQUATES.CLW (без предшествующего EVENT:). Мнозначный. Зависит от %Window.
%Control	Метки полей всех диалоговых элементов окна. Мнозначный. Зависит от %Window.
%ControlUse	USE переменные диалоговых элементов окна (не метки полей). Зависит от %Control.
%ControlStatement	Оператор объявления диалогового элемента (со всеми атрибутами). Зависит от %Control.

%ControlType	Тип диалогового элемента (MENU, ITEM, ENTRY, BUTTON, и т. д.). Зависит от %Control.
%ControlTemplate	Имя диалогового шаблона, разместившего в окне диалоговый элемент. Зависит от %Control.
%ControlTool	Содержит 1, если диалоговый элемент размещен на TOOLBAR. Зависит от %Control.
%ControlMenu	Содержит 1, если диалоговый элемент размещен в MENUBAR. Зависит от %Control.
%ControlToolBar	Содержит оператор объявления структуры TOOLBAR (вместе со всеми атрибутами), если диалоговый элемент является первым элементом TOOLBAR. Зависит от %Control.
%ControlMenuBar	Содержит оператор объявления структуры MENUBAR (вместе со всеми атрибутами), если диалоговый элемент является первым элементом MENUBAR. Зависит от %Control.
%ControlIndent	Уровень отступа оператора объявления диалогового элемента в генерируемой структуре данных. Зависит от %Control.
%ControlInstance	Номер экземпляра диалогового шаблона, разместившего в окне диалоговый элемент. Зависит от %Control.
%ControlOriginal	Первоначальная метка поля диалогового элемента, как она была указана в диалоговом шаблоне. Зависит от %Control.
%ControlFrom	Атрибут FROM диалогового элемента LIST или COMBO. Зависит от %Control.
%ControlAlert	Все атрибуты ALRT диалогового элемента. Многозначный. Зависит от %Control.
%ControlEvent	Все зависящие от поля события для диалогового элемента, как перечислены в файле EQUATES.CLW (без предшествующего EVENT:). Многозначный. Зависит от %Control.
%ControlField	Все переменные, размещенные в диалоговом элементе LIST, COMBO или SPIN. Многозначный. Зависит от %Control.

%ControlFieldHasIcon	Содержит 1, если поле в элементе LIST или COMBO отформатировано для вывода пиктограмм. Зависит от %ControlField.
%ControlFieldHasColor	Содержит 1, если поле в элементе LIST или COMBO отформатировано для раскрашивания. Зависит от %ControlField.
%ControlFieldHasTree	Содержит 1, если поле в элементе LIST или COMBO отформатировано для вывода дерева. Зависит от %ControlField.
%ControlFieldHasLocator	Содержит 1, если поле в элементе LIST или COMBO отформатировано, как локатор. Зависит от %ControlField.
%ControlFieldPicture	Содержит формирующий шаблон поля в элементе LIST или COMBO. Зависит от %ControlField.
%ControlFieldHeading	Содержит текст заголовка колонки в элементе LIST или COMBO. Зависит от %ControlField.

Символы элементов данных документа

%Report	Метка документа процедуры. Зависит от %Procedure.
%ReportStatement	Оператор объявления REPORT (со всеми атрибутами). Зависит от %Report.
%ReportControl	Метки полей для всех элементов данных документа. Мнозначный. Зависит от %Report.
%ReportControlUse	USE переменная элемента данных (не метка поля). Зависит от %ReportControl.
%ReportControlStatement	Оператор объявления элемента данных (со всеми атрибутами). Зависит от %ReportControl.
%ReportControlType	Тип (MENU, ITEM, ENTRY, BUTTON, ит.д.). Зависит от %ReportControl.
%ReportControlTemplate	Имя диалогового шаблона, разместившего в документе элемент данных. Зависит от %ReportControl.

%ReportControlIndent	Уровень отступа оператора объявления элемента данных в генерируемой структуре данных. Зависит от %ReportControl.
%ReportControlInstance	Номер экземпляра диалогового шаблона, разместившего в документе элемент данных. Зависит от %ReportControl.
%ReportControlOriginal	Первоначальная метка поля элемента данных, как она была указана в диалоговом шаблоне. Зависит от %ReportControl.
%ReportControlLabel	Метка элемента данных STRING документа. Зависит от %ReportControl.
%ReportControlField	Все переменные, размещенные в элементе данных LIST, COMBO или SPIN. Многозначный. Зависит от %ReportControl.

Символы формул

%Formula	Метка переменной результата для всех формул. Многозначный. Зависит от %Procedure.
%FormulaDescription	Описание формулы.
%FormulaClass	Идентификатор, определяющий местоположение формулы в сгенерированном тексте.
%FormulaInstance	Номер экземпляра диалогового шаблона для формулы, класс которой был указан в диалоговом шаблоне.
%FormulaExpression	Выражение для оценки условия или вычисления результата для всех формул. Многозначный. Зависит от %Formula.
%FormulaExpressionType	Содержит =, IF, ELSE, CASE или OF. Зависит от %FormulaExpression.
%FormulaExpressionTrue	Содержит номер строки с выражением для случая Истина в генерируемой формуле. Зависит от %FormulaExpression.
%FormulaExpressionFalse	Содержит номер строки с выражением для случая Ложь в генерируемой формуле. Зависит от %FormulaExpression.
%FormulaExpressionOf	Содержит номер строки с выражением для случая OF в генерируемой формуле. Зависит от %FormulaExpression.

%FormulaExpressionCase Содержит номер строки с выражением для присваивания в генерируемой формуле. Зависит от %FormulaExpression.

Символы файловой схемы

%Primary Метка главного файла, указанного в файловой схеме процедуры или использованного в ней диалогового шаблона.

%PrimaryKey Метка ключа для обращения к главному файлу. Зависит от %Primary.

%PrimaryInstance Номер экземпляра диалогового шаблона, для которого файл является главным. Зависит от %Primary.

%Secondary Метки всех присоединенных файлов, перечисленных в файловой схеме для процедуры или использованного в ней диалогового шаблона. Многозначный. Зависит от %Primary.

%SecondaryTo Метка присоединенного или главного файла, с которым связан присоединенный файл (файл “над ним”, как они выводятся в файловой схеме). Зависит от %Secondary.

%SecondaryType Содержит 1:MANY or MANY:1. Зависит от %Secondary.

%OtherFiles Метки всех перечисленных в разделе Other Data файлов процедуры. Многозначный.

Символы файлового драйвера

%Driver Имена всех зарегистрированных файловых драйверов.

%DriverDLL Имя .DLL файла драйвера. Зависит от %Driver.

%DriverLIB Имя .LIB файла драйвера. Зависит от %Driver.

%DriverDescription Описание файлового драйвера. Зависит от %Driver.

%DriverCreate Содержит 1, если драйвер поддерживает атрибут CREATE. Зависит от %Driver.

%DriverOwner	Содержит 1, если драйвер поддерживает атрибут OWNER. Зависит от %Driver.
%DriverEncrypt	Содержит 1, если драйвер поддерживает атрибут ENCRYPT. Зависит от %Driver.
%DriverReclaim	Содержит 1, если драйвер поддерживает атрибут RECLAIM. Зависит от %Driver.
%DriverMaxKeys	Максимальное количество ключей, которые поддерживает драйвер для каждого файла. Зависит от %Driver.
%DriverUniqueKey	Содержит 1, если драйвер поддерживает атрибут UNIQUE (нет атрибута DUP) для ключей. Зависит от %Driver.
%DriverRequired	Содержит 1, если драйвер поддерживает атрибут RECLAIM. Зависит от %Driver.
%DriverMemo	Содержит 1, если драйвер поддерживает MEMO поля. Зависит от %Driver.
%DriverBinMemo	Содержит 1, если драйвер поддерживает атрибут BINARY для MEMO полей. Зависит от %Driver.
%DriverSQL	Содержит 1, если драйвер является SQL-драйвером. Зависит от %Driver.
%DriverType	Все поддерживаемые драйвером типы данных. Многозначный. Зависит от %Driver.
%DriverOpcode	Все поддерживаемые драйвером операции. Многозначный. Зависит от %Driver.

Прочие символы

%ConditionalGenerate	Содержит 1, если в диалоге Application Options был включен переключатель Conditional Generation.
%CWVersion	Содержит номер релиза текущей версии Clarion for Windows. Например, для первого релиза версии 2.0 содержит 2000.
%Null	Ничего не содержит. Используется в сравнениях для

обнаружения пустых значений.

%True	Содержит 1.
%False	Содержит пустую строку (‘’).
%EOF	Содержит значение, которое отмечает конец файла при чтении его оператором #READ.
%BytesOutput	Содержит количество байт, записанных в текущий выходной файл. Этим можно пользоваться для обнаружения пустых точек вставки (если ничего не записано, то она пуста).
%EmbeddedID	Содержит символ идентифицирующий текущую точку вставки.
%Embedded Description	Содержит описание текущей точки вставки.
%EmbeddedParametrs	Содержит идентификатор экземпляра текущей точки вставки в виде разделенного запятыми списка.

Аннотированные примеры

Процедурный шаблон: Window

```
%StandardWindowCode #GROUP  
%StandardWindowHandling #GROUP  
“%StandardAcceptHandling #GROUP  
%StandardControlHandling #GROUP
```

Текстовый шаблон: ControlValueValidation

```
%CodeTPLValidationCode #GROUP
```

Control Template: DOSFileLookup

Распределенный шаблон: DateTimeDisplay

```
%DateTimeDisplayCode #GROUP
```

Процедурный шаблон: Window

Процедурный шаблон Window является родовым шаблоном, который лежит в основе любой процедуры управления окном. Так как большинство (если не все) процедуры в приложении Windows имеют окно, текст, который генерирует эта процедура, образует базис сгенерированного текста для большинства процедур.

Шаблон Window является также основным шаблоном на котором строятся другие процедурные шаблоны. Например, шаблон Browse в действительности является шаблоном Window с предварительно размещенными в этой процедуре диалоговыми шаблонами BrowseBox и BrowseUpdateButtons.

Следующий текст на языке шаблонов является полным текстом шаблона Window.

ЗАМЕЧАНИЕ: в этом и всех других примерах в этой книге символы продолжения строки шаблона (%) применяются для разделения слишком длинных строк текста, чтобы они могли поместиться на странице. В файлах шаблонов на диске эти символы не используются (и не должны использоваться) для продолжения строки текста шаблона; здесь они применяются для удобства.

```
#PROCEDURE(Window,'Generic Window
Handler'),WINDOW,HLP(~TPLProcWindow)
#LOCALDATA
LocalRequest          LONG,AUTO
OriginalRequest       LONG,AUTO
LocalResponse         LONG,AUTO
WindowOpened         LONG,AUTO
WindowInitialized     LONG
ForceRefresh          LONG,AUTO
#ENDLOCALDATA
#CLASS('Procedure Setup','Upon Entry into the Procedure')
#CLASS('Before Lookups','Refresh Window ROUTINE, before lookups')
#CLASS('After Lookups','Refresh Window ROUTINE, after lookups')
#CLASS('Procedure Exit','Before Leaving the Procedure')
#PROMPT('&Parameters:',@s255),%Parameters
#ENABLE(%ProcedureType='FUNCTION')
#PROMPT('Return Value:',FIELD),%ReturnValue
#ENDENABLE
#PROMPT('Window      Operation      Mode:',DROP('Use      WINDOW
setting|Normal|MDI|Modal')) %|
```

,%WindowOperationMode

```
#ENABLE(%INIActive)
  #BOXED('INI File Settings')
    #PROMPT('Save and Restore Window Location',CHECK) %|
    ,%INISaveWindow,DEFAULT(1),AT(10,,150)
  #ENDBOXED
#ENDENABLE
#AT(%CustomGlobalDeclarations)
  #INSERT(%StandardGlobalSetup)
#ENDAT
#INSERT(%StandardWindowCode)
```

Этот текст начинается с оператора `#PROCEDURE`. Этот оператор именуется процедурный шаблон и указывает, что он будет использовать структуру `WINDOW` (или `APPLICATION`), но не `REPORT`.

Секция `#LOCALDATA` определяет шесть локальных переменных, которые генерируются автоматически, как часть процедуры. Это общие локальные переменные для большинства генерируемых процедур.

Оператор `#CLASS` определяет классы формул процедуры для Редактора Формул. Они определяют, где в исходном тексте генерируются формулы.

Операторы `#PROMPT` создают диалоговые элементы в окне `Procedure Properties`. Первый позволяет программисту указать параметры, передаваемые в процедуру. Структура `#ENABLE` включает свой `#PROMPT` только если символ `%ProcedureType` содержит "FUNCTION". Это имеет место только, если в прототипе процедуры в поле `Prototype` (стандартное поле для всех процедур) указан тип возвращаемого данного.

Следующий `#PROMPT` позволяет программисту переопределить заданный в структуре `WINDOW` режим работы окна. Следующая структура `#ENABLE` активирует свои `#BOXED #PROMPT` только, если символ `%INIActive` содержит значение. Этот символ происходит от переключателя в окне `Global Settings`.

Структура `#AT` вызывает `%StandardGlobalSetup #GROUP`. В ней содержится код, определяющий использование в процедуре каких либо `.VBX` элементов. Если используются `.VBX`, то они добавляются в список файлов, которые должны поставляться вместе с приложением. Список выводится в файл `ProgramName.SHP`.

Вы должны обратить внимание, что ни один из этих операторов не генерирует какой-либо исходный (Clarion) текст. Оператор `#INSERT` размещает текст генерируемый в `%StandardWindowCode #GROUP` в конце этих операторов. Эта `#GROUP`

управляет генерацией всего шаблонного текста.

%StandardWindowCode #GROUP

Именно эта #GROUP генерирует весь исходный текст шаблона Window. Она включает все объявления локальных данных, стандартный текст управления окном и обеспечивает все “зацепки” для присоединения к сгенерированной процедуре всех диалоговых и распределенных шаблонов.

```
#GROUP(%StandardWindowCode)
#IF(NOT %Window)
    #ERROR(%Procedure & ' Error: No Window Defined!')
    #RETURN
#ENDIF
#DECLARE(%FirstField)
#DECLARE(%LastField)
#DECLARE(%ProgressWindowRequired)
#INSERT(%FieldTemplateStandardButtonMenuPrompt)
#INSERT(%FieldTemplateStandardEntryPrompt)
#INSERT(%FieldTemplateStandardCheckBoxPrompt)
#EMBED(%GatherSymbols,'Gather Template Symbols'),HIDE
#INSERT(%FileControlInitialize)
%Procedure %ProcedureType%Parameters
#FOR(%LocalData)
%[20]LocalData %LocalDataStatement
#ENDFOR
#INSERT(%StandardWindowGeneration)
#IF(%ProgressWindowRequired)
#INSERT(%StandardProgressWindow)
#ENDIF
    CODE
    PUSHBIND
    #EMBED(%ProcedureInitialize,'Initialize the Procedure')
    LocalRequest = GlobalRequest
    OriginalRequest = GlobalRequest
    LocalResponse = RequestCancelled
    ForceRefresh = False
    CLEAR(GlobalRequest)
    CLEAR(GlobalResponse)
    #EMBED(%ProcedureSetup,'Procedure Setup')
    IF KEYCODE() = MouseRight
        SETKEYCODE(0)
```

```

END
#INSERT(%StandardFormula,'Procedure Setup')
#INSERT(%FileControlOpen)
#INSERT(%StandardWindowOpening)
#EMBED(%PrepareAlerts,'Preparing Window Alerts')
#EMBED(%BeforeAccept,'Preparing to Process the Window')
#MESSAGE('Accept Handling',3)
ACCEPT
#EMBED(%AcceptLoopBeforeEventHandling, %|
      'Accept Loop, Before CASE EVENT() handling')
  CASE EVENT()
#EMBED(%EventCaseBeforeGenerated, %|
      'CASE EVENT() structure, before generated code')
  #INSERT(%StandardWindowHandling)
#EMBED(%EventCaseAfterGenerated, %|
      'CASE EVENT() structure, after generated code')
  END
#EMBED(%AcceptLoopAfterEventHandling, %|
      'Accept Loop, After CASE EVENT() handling')
  #SUSPEND
  #?CASE ACCEPTED()
  #INSERT(%StandardAcceptedHandling)
  #?END
  #RESUME
#EMBED(%AcceptLoopBeforeFieldHandling, %|
      'Accept Loop, Before CASE FIELD() handling')
  #SUSPEND
  #?CASE FIELD()
#EMBED(%FieldCaseBeforeGenerated, %|
      'CASE FIELD() structure, before generated code')
  #INSERT(%StandardControlHandling)
#EMBED(%FieldCaseAfterGenerated, %|
      'CASE FIELD() structure, after generated code')
  #?END
  #RESUME
#EMBED(%AcceptLoopAfterFieldHandling, |
      'Accept Loop, After CASE FIELD() handling')
END
DO ProcedureReturn
!-----
ProcedureReturn ROUTINE
!|
!| This routine provides a common procedure exit point for all

```

```

!| template generated procedures.
!|
!| First, all of the files opened by this procedure are closed.
!|
!| Next, if it was opened by this procedure, the window is closed.
!|
!| Next, GlobalResponse is assigned a value to signal the calling
!| procedure what happened in this procedure.
!|
!| Next, we replace the BINDings that were in place when the procedure
!| initialized (and saved with PUSHBIND) using POPBIND.
!|
#IF(%ReturnValue)
!| Finally, we return to the calling procedure, passing %ReturnValue
!| back.
#ELSE
!| Finally, we return to the calling procedure.
#ENDIF
!|
  #INSERT(%FileControlClose)
  #INSERT(%StandardWindowClosing)
  #EMBED(%EndOfProcedure,'End of Procedure')
  #INSERT(%StandardFormula,'Procedure Exit')
  IF LocalResponse
    GlobalResponse = LocalResponse
  ELSE
    GlobalResponse = RequestCancelled
  END
  POPBIND
  #IF(%ProcedureType='FUNCTION')
  RETURN(%ReturnValue)
  #ELSE
  RETURN
  #ENDIF
!-----
InitializeWindow ROUTINE
!|
!| This routine is used to prepare any control templates for use. It
!| should be called once per procedure.
!|
  #EMBED(%WindowInitializationCode,'Window Initialization Code')
  DO RefreshWindow
!-----

```


RefreshWindow ROUTINE

```

!|
!| This routine is used to keep all displays and control templates
!| current.
!|
IF %Window{Prop:AcceptAll} THEN EXIT.
#EMBED(%RefreshWindowBeforeLookup, %|
    'Refresh Window routine, before lookups')
    #INSERT(%StandardFormula,'Before Lookups')
    #INSERT(%StandardSecondaryLookups)
    #INSERT(%StandardFormula,'After Lookups')
#EMBED(%RefreshWindowAfterLookup, %|
    'Refresh Window routine, after lookups')
#EMBED(%RefreshWindowBeforeDisplay, %|
    'Refresh Window routine, before DISPLAY()')
    DISPLAY()
    ForceRefresh = False
!-----

```

SyncWindow ROUTINE

```

#EMBED(%SyncWindowBeforeLookup, %|
    'Sync Record routine, before lookups')
    #INSERT(%StandardFormula,'Before Lookups')
    #INSERT(%StandardSecondaryLookups)
    #INSERT(%StandardFormula,'After Lookups')
#EMBED(%SyncWindowAfterLookup,'Sync Record routine, after lookups')
!-----
#EMBED(%ProcedureRoutines,'Procedure Routines')
#EMBED(%LocalProcedures,'Local Procedures'),HIDE

```

Эта группа начинается с обязательного оператора #GROUP, который идентифицирует ее для ссылок в операторах #INSERT. Первый оператор #INSERT выполняет группу шаблонных операторов, которые выделяют примененные в процедуре диалоговые элементы .VBX, чтобы добавить их к списку используемых в приложении пользовательских диалоговых элементов. Так строится список .VBX файлов, которые должны быть включены в поставку конечному пользователю вместе с приложением.

Проверка #IF(NOT %Window) определяет не забыл ли программист определить окно для этой процедуры. Оператор #ERROR напоминает программисту об ошибке, а #RETURN немедленно прекращает всю дальнейшую генерацию текста процедуры. Оператор #DECLARE объявляет два шаблонных символа для внутреннего использования в #GROUP-ах, которые вызываются для генерации текста процедуры, и “флажок”, который определяет нужно ли для процедуры окно “продвижения”.

Следующие три оператора `#INSERT` вставляют `#GROUP`-ы, которые содержат структуры `#FIELD` для формирования стандартного ввода, который будет появляться в диалоге `Actions` для размещенных в окне процедуры диалоговых элементов `BUTTON`, `ENTRY` и `CHECK`.

Оператор `%GatherSymbols #EMBED` имеет атрибут `HIDE`. Это означает, что он не появится в списке доступных программисту точек вставки исходного текста, делая точку вставки доступной только для внутреннего использования (для размещения текста текстовыми, диалоговыми или распределенными шаблонами).

Оператор `#INSERT(%FileControlInitialize)` вставляет `#GROUP`, которая пополняет символы, ведущие учет использованных в приложении файлов задействованными в этой процедуре файлами.

Оператор `%Procedure %ProcedureType%Parameters` генерирует первый оператор исходного текста на языке Clarion процедуры. Он генерирует оператор `PROCEDURE` или `FUNCTION` со списком параметров или без него, как требуется.

Цикл `#FOR(%LocalData)` генерирует все объявления локальных переменных процедуры. Синтаксическая конструкция `%[20]LocalData` означает, что символ `%LocalData` расширяется пробелами, чтобы заполнить по крайней мере 20 позиций перед размещением символа `%LocalData`. Это выравнивает типы данных в каждом объявлении данных на колонку 22.

Оператор `#INSERT(%StandardWindowGeneration)` генерирует структуру данных `WINDOW` или `APPLICATION` процедуры. Эта `#GROUP` содержит также операторы `#EMBED`, которые позволяют программисту вставить текст перед и после оконной структуры.

Оператор `#IF(%ProgressWindowRequired)` условно вставляет (`#INSERT`) группу `%StandardProgressWindow`, которая генерирует для процедуры `WINDOW` структуру `ProgressWindow`.

Далее генерируется оператор `CODE`, начинающий раздел выполняемого текста процедуры, который начинается оператором `PUSHBIND` во избежание недоразумений, связанных с областью действия оператора `BIND`. Оператор `%ProcedureInitialize #EMBED` является первой доступной программисту точкой вставки исходного текста в секции исполняемого текста процедуры.

Следующие шесть операторов языка Clarion непосредственно генерируются в процедуру для подготовки к выполнению ее действий, как было указано процедуре с помощью переменной `GlobalRequest`. Оператор `%ProcedureSetup #EMBED` является следующей доступной программисту точкой вставки исходного текста в секции исполняемого текста процедуры.

Структура `IF KEYCODE() = MouseRight` проверяет не была ли процедура вызвана с

помощью правой кнопки мыши и всплывающего меню. Если это так, она обеспечивает очистку Keycode() во избежание повторной обработки.

Оператор %StandardFormula #INSERT генерирует все формулы класса “Procedure Setup”. Вслед за этим #INSERT(%FileControlOpen) генерирует текст открытия всех использованных в процедуре файлов (если они еще не открыты). Эта #GROUP также содержит два оператора #EMBED, которые позволяют программисту вставить исходный текст перед и после открытия файлов.

Оператор #INSERT(%StandardWindowOpening) генерирует оператор OPEN(window) и текст управления .INI файлом (если программист включил переключатель “Use .INI file to save and restore program settings”). Эта #GROUP также содержит два оператора #EMBED, которые позволяют программисту вставить исходный текст перед и после открытия окна.

Следующие два оператора #EMBED позволяют программисту вставить исходный текст перед входом в АСCEPT-цикл процедуры. Оператор #MESSAGE выводит свое сообщение в процессе генерации исходного текста.

Структура ACCEPT это Clarion-овский цикл обработки событий. Следующий оператор #EMBED (%AcceptLoopBeforeEventHandling) позволяет программисту вставить код с которого начнется исполнение каждого прохода цикла ACCEPT. Структура CASE EVENT(), содержащая весь текст для обработки независимых от поля событий, генерируется оператором #INSERT(%StandardWindowHandling). Эта #GROUP детально рассмотрена в следующем параграфе. Два оператора #EMBED, которые окружают оператор #INSERT, и #EMBED вслед за структурой CASE EVENT дают возможность программисту явно обработать любое не связанное с полем событие, которое не обслуживается сгенерированным текстом.

Оператор #SUSPEND означает, что условные операторы (те, которые предпосланы знаками #?) будут генерироваться, если для события будет сгенерирован явный оператор (без #?) или программист разместит некоторый исходный текст или использует текстовый шаблон в точке вставки, связанной с обрабатываемым событием. Это механизм, который позволяет языку шаблонов Clarion генерировать только тот текст, который необходим процедуре, исключая генерацию ненужных фрагментов.

Структура #?CASE ACCEPTED() содержит весь текст для обработки событий завершения пунктов меню. Так как пункты меню генерируют только событие Accepted, это предотвращает излишнее разрастание структуры CASE FIELD(). Эта строка текста, будучи предпосланной знаками #?, будет генерироваться только, если внутри нее будет генерироваться еще какой-либо текст, что исключает пустые структуры CASE. Текст структуры CASE генерируется оператором #INSERT(%StandardAcceptedHandling). Эта #GROUP также подробно рассматривается в отдельном параграфе. Оператор #?END будет генерировать оператор END только, если уже будет сгенерирован какой-нибудь текст.

Оператор `#RESUME` завершает структуру `#SUSPEND`. Если не сгенерировано ни какого текста, то не будет сгенерировано ни одного условного оператора исходного текста (предпосланного знаками `#?`) между `#SUSPEND` и `#RESUME`.

Структура `#?CASE FIELD()` (также заключенная в скобки `#SUSPEND` и `#RESUME`) содержит весь текст для обработки зависящих от поля событий. Текст структуры `CASE` генерируется оператором `#INSERT(%StandardControlHandling)` (окруженным парой своих операторов `#EMBED`). Эта `#GROUP` также подробно рассматривается в отдельном параграфе. Оператор `#?END` будет генерировать оператор `END` только, если уже будет сгенерирован какой-нибудь текст. Оператор `#EMBED`, непосредственно следующий за `#RESUME`, обеспечивает точку вставки в самом конце цикла `ACCEPT`.

Оператор `END` завершает цикл `ACCEPT`. Этот оператор всегда генерируется (как и `ACCEPT`), потому что для обработки событий в каждом окне требуется непосредственно связанный с ним `ACCEPT`-цикл. Оператор `DO ProcedureReturn` выполняет “заключительный текст” процедуры.

Подпрограмма `ProcedureReturn ROUTINE` начинается с блока комментариев, которые генерируются в `Clarion` текст и объясняют назначение подпрограммы. Первая строка текста подпрограммы, это оператор `#INSERT(%FileControlClose)`. Он генерирует текст для закрытия тех файлов, которые были открыты процедурой. Эта `#GROUP` также содержит два оператора `#EMBED`, которые позволяют программисту вставить исходный текст до и после закрытия файлов.

Оператор `#INSERT(%StandardWindowClosing)` генерирует оператор `CLOSE(window)` и текст управления файлом `.INI` (если программист включил переключатель “Use `.INI` file to save and restore program settings”). Эта `#GROUP` также содержит два оператора `#EMBED`, которые позволяют программисту вставить исходный текст до и после закрытия окна.

Следующий оператор `#EMBED` позволяет программисту вставить исходный текст перед закрытием окна процедуры. Следующий оператор `#INSERT` генерирует все формулы класса “`Procedure Exit`”. Следующие пять операторов языка `Clarion` формируют сигнал вызвавшей процедуре о выполненном действии, передавая его через переменную `GlobalResponse`. Оператор `POPBIND` исключает недоразумения, связанные с областью действия оператора `BIND`. Затем структура `#IF` проверяет не является ли процедура функцией и генерирует надлежащий оператор возврата (`RETURN`).

Подпрограмма `InitializeWindow ROUTINE` является стандартной подпрограммой для всех поставляемых шаблонов `Clarion`. Она начинается с блока комментариев, а следующий оператор `#EMBED` позволяет программисту выполнить любой собственный

инициализирующий текст и обеспечивают текстовым, диалоговым и распределенным шаблонам возможность размещения их текста инициализации окна. Оператор `DO RefreshWindow` вызывает подпрограмму вывода текущего содержимого `USE`-переменных для всех диалоговых элементов в момент инициализации окна.

Подпрограмма `RefreshWindow ROUTINE` это другая стандартная подпрограмма для всех поставляемых шаблонов `Clarion`, которая обеспечивает связи файлов `MANY:1` и обновляет окно, чтобы изменившиеся данные были правильно показаны пользователю в любой момент времени. Эта `ROUTINE` начинается с блока комментариев и оператора `IF %Window{Prop:AcceptAll} THEN EXIT`. Эта проверка предотвращает ненужную перерисовку экрана, если `ACCEPT` цикл находится в “non-stop” режиме проверки правильности данных перед их записью на диск.

Первый оператор `#EMBED` позволяет программисту вставить текст перед обращением к связанным файлам. Следующий оператор `#INSERT` генерирует все формулы класса “Before Lookups”, затем оператор `#INSERT(%StandardSecondaryLookups)` генерирует текст для обращения ко всем связанным записям в процедуре. Следующий оператор `#INSERT` генерирует все формулы класса “After Lookups”, затем следует оператор `#EMBED`, чтобы позволить программисту вставить текст после обращения к связанным файлам. Оператор `DISPLAY` выводит любые изменившиеся данные на экран, а `ForceRefresh = False` сбрасывает флажок обновления экрана.

Подпрограмма `SyncWindow ROUTINE` также является стандартной подпрограммой для всех поставляемых шаблонов `Clarion`. Она выполняет те же самые обращения к связанным записям, что и `the RefreshWindow ROUTINE` с аналогичными точками вставки, но не обновляет экран. Вместо этого она обеспечивает, что все буферы записей содержат правильные данные. Эта `ROUTINE` обычно вызывается перед выполнением некоторых действий, которые могут потребовать данные из текущей подсвеченной записи в структуре `LIST`.

Предпоследний оператор `#EMBED` позволяет программисту добавить любые подпрограммы `ROUTINE`, которые он вызывал в своих текстах в точках вставки. Последний оператор `#EMBED` позволяет всем остальным шаблонам размещать здесь свои локальные процедуры и функции.

%StandardWindowHandling #GROUP

Эта `#GROUP` генерирует весь необходимый текст для обработки в процедуре независимых от поля событий. Она генерирует свой текст в структуру `CASE EVENT()` шаблона `Window`.

`#GROUP(%StandardWindowHandling)`

```
#FOR(%WindowEvent)
  #SUSPEND
#?OF EVENT:%WindowEvent
  #EMBED(%WindowEventHandling,'Window Event Handling'),%WindowEvent
  #CASE(%WindowEvent)
  #OF('OpenWindow')
IF NOT WindowInitialized
  DO InitializeWindow
  WindowInitialized = True
END
  #IF(%FirstField)
  SELECT(%FirstField)
  #ENDIF
  #OF('GainFocus')
ForceRefresh = True
IF NOT WindowInitialized
  DO InitializeWindow
  WindowInitialized = True
ELSE
  DO RefreshWindow
END
  #OF('Sized')
ForceRefresh = True
  DO RefreshWindow
  #ENDCASE
#EMBED(%PostWindowEventHandling, %|
      'Window Event Handling, after generated code'), %|
      %WindowEvent
  #RESUME
#ENDFOR
OF Event:Rejected
  #EMBED(%WindowEventHandlingBeforeRejected, %|
      'Window Event Handling - Before Rejected')

  BEEP
  DISPLAY(?)
  SELECT(?)
  #EMBED(%WindowEventHandlingAfterRejected, %|
      'Window Event Handling - After Rejected')
#SUSPEND
#?ELSE
  #EMBED(%WindowOtherEventHandling,'Other Window Event Handling')
#RESUME
```

Эта группа начинается с обязательного оператора `#GROUP`, который идентифицирует ее для ссылок в операторах `#INSERT`.

Оператор `#SUSPEND` начинает секцию, которая будет условно генерировать текст только, если будут сгенерированы безусловные (без `#?`) операторы исходного текста, или программист разместит исходный текст или использует текстовый шаблон в одной из точек вставки.

Оператор `#?OF EVENT:%WindowEvent` условно генерирует вариант `OF` структуры `CASE EVENT()` для текущего обрабатываемого экземпляра `%WindowEvent`. Так как эта строка текста предпослана знаками `#?`, будет генерироваться только, если в ней будет сгенерирован какой-либо другой текст, исключая тем-самым пустой вариант `OF`.

Оператор `#EMBED` является ключом к процессу генерации исходного текста и к взаимодействию с текстовыми, диалоговыми и распределенными шаблонами. Благодаря тому, что в конце его имеется `“,%WindowEvent”`, программист получит отдельную точку вставки текста для каждого экземпляра символа `%WindowEvent`. Это означает, что программист может написать свой текст для любого, независимого от поля события. Это означает также, что любой текстовый, диалоговый или распределенный шаблон, который программист использовал в процедуре, может, при необходимости, генерировать текст в эти точки вставки, чтобы сформировать текст, необходимый для выполнения их функций.

Структура `#CASE(%WindowEvent)` генерирует безусловный исходный текст для независящих от поля событий. Вариант `#OF(‘OpenWindow’)` проверяет событие `EVENT:OpenWindow` и генерирует отметку в переменной `WindowInitialized` для условной инициализации окна. Этот текст выполняется, если до этого не было событий `EVENT:GainFocus` (таких как открытие второго окна в том же исполняемом процессе, который имеет текущий фокус). Оператор `SELECT(%FirstField)` генерируется только, если в окне есть диалоговые элементы, которые могут получать фокус.

Оператор `#OF(‘GainFocus’)` проверяет событие `EVENT:GainFocus` и генерирует оператор `ForceRefresh = True` и затем проверяет инициализировано ли окно (это нужно при переключении пользователя между активными процессами). Если окно не инициализировано, оно инициализируется, иначе просто обновляется. Оператор `#OF(‘Sized’)` проверяет событие `EVENT:Sized` и генерирует `ForceRefresh = True` и `DO RefreshWindow` для обновления окна после того, как пользователь изменил его размеры.

Оператор `#ENDCASE` завершает структуру `#CASE`. Последующий оператор `#EMBED` дает возможность программисту ввести после сгенерированного текста свой собственный текст для обработки любого из этих событий. Оператор `#RESUME` завершает секцию `#SUSPEND`. Если не было сгенерировано никакого текста, то ни один оператор условного текста (предпосланный `#?`) между `#SUSPEND` и `#RESUME` не

будет сгенерирован.

Оператор **#ENDFOR** завершает цикл **#FOR**, затем **OF EVENT:Rejected** создает две точки вставки - до и после стандартного текста, который предупреждает пользователя, что ввод данных в текущее поле отвергнут (обычно из-за выхода за границы диапазона) и оставляет его в том же самом поле.

Оператор **#SUSPEND** начинает новую секцию условной генерации. Это означает, что оператор **#?ELSE** генерирует **ELSE** только, если оператором **#EMBED** генерируется исходный текст. Оператор **#RESUME** завершает эту секцию **#SUSPEND**.

“%StandardAcceptHandling #GROUP

Эта **#GROUP** генерирует весь текст для обработки событий завершения пунктов меню. Она генерирует свой текст в структуру **CASE ACCEPTED()** шаблона **Window**.

```
#GROUP(%StandardAcceptedHandling)
#FOR(%Control),WHERE(%ControlMenu)
    #FIX(%ControlEvent,'Accepted')
    #MESSAGE('Control Handling: ' & %Control,3)
    #SUSPEND
#?OF %Control
    #EMBED(%ControlPreEventHandling, %|
        'Control Event Handling, before generated code'), %|
        %Control,%ControlEvent
    #INSERT(%FieldTemplateStandardHandling)
    #EMBED(%ControlEventHandling,'Internal Control Event Handling'), %|
        %Control,%ControlEvent,HIDE
    #EMBED(%ControlPostEventHandling, %|
        'Control Event Handling, after generated code'), %|
        %Control,%ControlEvent
    #RESUME
#ENDFOR
```

Этот текст начинается с оператора цикла **#FOR(%Control),WHERE(%ControlMenu)**. Атрибут **WHERE** ограничивает исполнение цикла только теми экземплярами **%Control**, которые содержат пункты меню. Оператор **#FIX** гарантирует, что цикл будет иметь дело только с событиями **Accepted**.

Оператор **#MESSAGE** выводит свое сообщение в процессе генерации. Оператор **#SUSPEND** начинает секцию условной генерации текста.

Оператор **#?OF %Control** условно генерирует вариант **OF** структуры **CASE ACCEPTED()** для текущего обрабатываемого экземпляра **%Control**. Эта строка текста, так

как она предпослана знаками #?, будет генерироваться только, если в ней будет сгенерирован какой-либо другой текст, исключая, тем самым, генерацию пустого варианта OF.

Все следующие три оператора #EMBED в конце имеют атрибуты “,%Control,%ControlEvent”, поэтому программист получит отдельные точки вставки для каждого экземпляра символа %ControlEvent для каждого экземпляра символа %Control. В данном случае это будут только события Accepted.

Оператор #INSERT(%FieldTemplateStandardHandling) генерирует текст для обработки всего ввода программиста, который он сделал во всех диалогах Actions для диалоговых элементов окна. Следующие два оператора #EMBED тоже имеют в конце атрибуты “,%Control,%ControlEvent”, Первый из них имеет атрибут HIDE, поэтому он доступен только для текстовых, диалоговых и распределенных шаблонов. Эти три оператора #EMBED дают программисту точки вставки как до, так и после текста, сгенерированного на основании параметров, заданных на страничке Actions.

#RESUME завершает секцию #SUSPEND. #ENDFOR завершает цикл по %Control.

%StandardControlHandling #GROUP

Эта #GROUP генерирует весь текст, необходимый для обработки в процедуре зависящих от поля событий. Она генерирует свой текст в структуру CASE FIELD() шаблона Window.

```
#GROUP(%StandardControlHandling)
#FOR(%Control),WHERE(%Control)
  #MESSAGE('Control Handling: ' & %Control,3)
  #SUSPEND
#?OF %Control
  #EMBED(%ControlPreEventCaseHandling, %|
    'Control Handling, before event handling'), %|
    %Control
  #?CASE EVENT()
    #IF(NOT %ControlMenu)
      #FOR(%ControlEvent)
        #SUSPEND
      #?OF EVENT:%ControlEvent
        #EMBED(%ControlPreEventHandling, %|
          'Control Event Handling, Before Generated Code'),%|
          %Control,%ControlEvent
        #INSERT(%FieldTemplateStandardHandling)
      #EMBED(%ControlEventHandling, %|
        'Internal Control Event Handling'), %|
        %Control,%ControlEvent,HIDE
      #EMBED(%ControlPostEventHandling, %|
```

```

        'Control Event Handling, After Generated Code'), %|
        %Control,%ControlEvent
    #RESUME
#ENDFOR
#ELSE
#?OF EVENT:Accepted
#ENDIF
#SUSPEND
#?ELSE
#EMBED(%ControlOtherEventHandling, %|
        'Other Control Event Handling'),%Control
#RESUME
#?END
#EMBED(%ControlPostEventCaseHandling, %|
        'Control Handling, after event handling'), %|
        %Control
#RESUME
#ENDFOR

```

Этот текст начинается с цикла `#FOR(%Control),WHERE(%Control)`. Вариант `WHERE` может показаться на первый взгляд излишним, так как `#FOR` будет перебирать только существующие экземпляры `%Control`. Однако, так как некоторым диалоговым элементам не нужны метки поля, могут быть допустимые экземпляры `%Control`, которые не содержат собственно значений `%Control`. Следовательно, атрибут `WHERE` ограничивает этот цикл `#FOR` теми экземплярами `%Control`, которые в действительности содержат метку поля для диалогового элемента.

Оператор `#MESSAGE` выводит свое сообщение в процессе генерации. Оператор `#SUSPEND` начинает секцию условной генерации текста.

Оператор `#?OF %Control` условно генерирует вариант `OF` структуры `CASE FIELD()` для текущего обрабатываемого экземпляра `%Control`. Эта строка текста, так как она предпослана знаками `#?`, будет генерироваться только, если в ней будет сгенерирован какой-либо другой текст, исключая, тем самым, генерацию пустого варианта `OF`.

Первый оператор `#EMBED` позволяет программисту справиться с любой ситуацией, когда требуются действия до любого автоматически сгенерированного текста. Оператор `#?CASE EVENT()` условно генерирует для диалогового элемента структуру `CSE EVENT()`. Оператор `#IF(NOT %ControlMenu)` отфильтровывает все пункты меню, так как они обрабатываются в `%StandardAcceptedHandling #GROUP`. Цикл `#FOR(%ControlEvent)` перебирает все возможные события, которые может создавать обрабатываемый диалоговый элемент.

Оператор `#SUSPEND` начинает другую, вложенную в предыдущую, секцию условной генерации. Это создает несколько уровней условной генерации исходного текста. Внешняя секция автоматически генерируется, если во внутренней секции генерируется какой-либо текст.

Оператор `#?OF EVENT:%ControlEvent` условно генерирует вариант `OF` структуры `CASE EVENT()` для текущего обрабатываемого экземпляра `%Control`. Поскольку эта строка текста предпослана символами `#?`, то она будет генерироваться только, если в ней будет сгенерирован какой-либо текст, исключая, тем самым, генерацию пустого варианта `OF`.

Следующий оператор `#EMBED` в конце имеет `“,%Control,%ControlEvent”`, поэтому программист получит отдельные точки вставки для каждого экземпляра символа `%ControlEvent` каждого экземпляра символа `%Control`. Это означает, что программист может написать свой текст для любого связанного с полем события любого диалогового элемента. Это также означает, что любой текстовый, диалоговый или распределенный шаблон, который программист разместил в процедуре, может, при необходимости, генерировать текст в эти точки вставки, чтобы сформировать текст, необходимый для выполнения его функций. Эти точки вставки используются в операторах `#AT` в текстовых, диалоговых или распределенных шаблонах.

Оператор `#INSERT(%FieldTemplateStandardHandling)` генерирует текст для обработки всего ввода программиста, который он сделал во всех диалогах `Actions` для диалоговых элементов окна. Приглашения в диалогах `Actions` задаются в структурах `#FIELD`, которые были вставлены (`#INSERT`) в начале шаблона `Window`.

Следующие два оператора `#EMBED` также содержат в конце `“,%Control,%ControlEvent”`, поэтому программист получит отдельные точки вставки для каждого экземпляра символа `%ControlEvent` каждого экземпляра символа `%Control`. Первый из них имеет атрибут `HIDE` и поэтому доступен только для использования в текстовых, диалоговых и распределенных шаблонах. Эти три оператора `#EMBED` дают программисту точки вставки текста как перед, так и после любого автоматически сгенерированного в них с помощью элементов ввода в диалогах `Actions` текста.

Оператор `#RESUME` завершает внутреннюю секцию условной генерации, а `#ENDFOR` завершает цикл по `%ControlEvent`. `#ELSE` относится к предшествующему `#IF(NOT %ControlMenu)` и будет генерировать пустой `OF EVENT:Accepted` с последующим `ELSE` для пунктов меню, если программист разместит свой текст в точке `OtherControlEventHandling`. Это исключает дублирование текста для событий меню `EVENT:Accepted`, позволяя в то же время обрабатывать для них любые пользовательские события.

Оператор `#SUSPEND` начинает другую вложенную секцию условной генерации. Это означает, что оператор `#?ELSE` генерирует `ELSE` только, если оператором `#EMBED`

будет сгенерирован какой-либо текст. Оператор #RESUME завершает эту секцию #SUSPEND.

Оператор #?END генерирует оператор END для структуры CASE FIELD(), если был сгенерирован какой-либо текст, затем оператор #RESUME завершает внешнюю секцию #SUSPEND. Оператор #ENDFOR завершает цикл по %Control.

Текстовый шаблон: *ControlValueValidation*

Текстовый шаблон ControlValueValidation выполняет проверку введенных данных для вводных диалоговых элементов ENTRY, SPIN или COMBO, отыскивая введенное пользователем значение в другом файле данных. Если поиск оказывается успешным, то введенное значение допустимо. Если нет, то вызывается процедура, чтобы дать возможность пользователю выбрать правильное значение из справочного файла.

Этот текстовый шаблон предназначен для генерации текста в точки вставки EVENT:Selected или EVENT:Accepted диалоговых элементов ENTRY, SPIN или COMBO. Это диалоговые элементы в которые пользователь может непосредственно вводить данные.

```
#CODE(ControlValueValidation,'Control Value Validation')
#RESTRICT
  #CASE(%ControlType)
  #OF('ENTRY')
  #OROF('SPIN')
  #OROF('COMBO')
    #CASE(%ControlEvent)
    #OF('Accepted')
    #OROF('Selected')
      #ACCEPT
    #ELSE
      #REJECT
    #ENDCASE
  #ELSE
    #REJECT
  #ENDCASE
#ENDRESTRICT
#DISPLAY('This Code Template is used to perform a control value')
#DISPLAY('validation. This Code Template only works for')
#DISPLAY('the Selected or Accepted Events for an Entry Control.')
#DISPLAY('')
#PROMPT('Lookup Key',KEY),%LookupKey,REQ
#PROMPT('Lookup Field',COMPONENT),%LookupField,REQ
#PROMPT('Lookup Procedure',PROCEDURE),%LookupProcedure
```

```

#DISPLAY('')
#DISPLAY('The Lookup Key is the key used to perform the value valida-
tion.')
```

```

#DISPLAY('If the Lookup Key is a multi-component key, you must insure that')
#DISPLAY('other key elements are primed BEFORE this Code Template is used.')
```

```

#DISPLAY('')
#DISPLAY('The Lookup field must be a component of the Lookup Key. Before)
#DISPLAY('execution of the lookup code, this field will be assigned the value of)
#DISPLAY('the control being validated, and the control will be assigned the value)
#DISPLAY('of the lookup field if the Lookup procedure is successful.')
```

```

#DISPLAY('')
#DISPLAY('The Lookup Procedure is called to let the user to select a value.
)
#DISPLAY('Request upon entrance to the Lookup will be set to SelectRecord,
and )
#DISPLAY('successful completion is signalled when Response = RequestCompleted.')
```

```

#IF(%ControlEvent='Accepted')
IF %Control{PROP:Req} = False AND NOT %ControlUse #<! If not required and
empty
ELSE
#INSERT(%CodeTPLValidationCode)
END
#ELIF(%ControlEvent='Selected')
#INSERT(%CodeTPLValidationCode)
#ELSE
#ERROR('This Code Template must be used for Accepted or Selected Events!')
```

```

#ENDIF
```

Текстовый шаблон всегда начинается с оператора **#CODE**, который идентифицирует его в наборе шаблонов и создает описание, которое появляется в списке доступных текстовых шаблонов для данной точки вставки.

Структура **#RESTRICT** определяет в каких точках вставки в качестве одного из вариантов выбора будет появляться текстовый шаблон. Структура **#CASE(%ControlType)** ограничивает точки вставки диалоговыми элементами **ENTRY**, **SPIN** и **COMBO**, а структура **#CASE(%ControlEvent)** ограничивает точки вставки событиями **EVENT:Accepted** и **EVENT:Selected**.

Оператор **#АССЕПТ** указывает, что это подходящие точки вставки, в то время как **#РЕ-ЖЕСТ** указывает, что точки вставки всех остальных диалоговых элементов и других событий не подходят для появления текстового шаблона в качестве элемента выбора.

Все операторы **#DISPLAY** выводят программисту свой текст в диалоге текстовых


```

file
  IF ERRORCODE()                                #<! IF record not
found                                           #<! Set Action for Lookup
  GlobalRequest = SelectRecord                  #<! Call Lookup
  %LookupProcedure
Procedure
  LocalResponse = GlobalResponse                #<! Save Returned
Action
  GlobalResponse = RequestCancelled             #<! Clear the Action Value
  IF LocalResponse = RequestCompleted          #<! IF Lookup successful
    %ControlUse = %LookupField                 #<! Move value to
control field
    #IF(%ControlEvent='Accepted')              #! IF a Post-Edit Validation

    ELSE                                       #<! ELSE (IF Lookup
NOT...)
    SELECT(%Control)                          #<! Select the control
    CYCLE                                     #<! Go back to
ACCEPT
    #ENDIF                                    #! END (IF a Pre-
Edit...)
    END                                       #<! END (IF Lookup
successful)
    #IF(%ControlEvent='Selected')             #! IF a Pre-Edit Validation
    SELECT(%Control)                          #<! Select the control
    #ENDIF                                    #! END (IF a Pre-
Edit...)
    END                                       #<! END (IF record
not found)

```

Эта #GROUP начинается с генерации присваивания %LookupField = %ControlUse. Эта присваивает USE переменную диалогового элемента полю, указанному во втором вводимом поле, а именно: поле ключа, которое должно содержать правильное значение.

Оператор #FIND(%Field,%LookupField) просматривает все поля в словаре данных, отыскивая поле соответствующее содержащемуся в %LookupField. Это фиксирует %Field и %File на правильные значения для генерации оператора GET(%File,%LookupKey). Это принимает форму GET(file,key) оператора GET для извлечения единственной записи справочного файла с соответствующим значением ключевого поля.

Структура IF ERRORCODE() проверяет успешность выполнения операции GET. При возникновении ошибки поиск оказался неудачным и оператор GlobalRequest = SelectRecord подготавливает вызов процедуры выбора из справочника, который осуществляется

оператором %LookupProcedure.

После возврата из процедуры выбора из справочника LocalResponse = GlobalResponse передает процедуре код возврата. Затем оператор GlobalResponse = RequestCancelled очищает глобальную переменную с тем, чтобы другой параллельный процесс не мог получить неправильное значение. Это должно быть сделано немедленно, до того, как пользователь будет иметь шанс переключиться на другой процесс.

Структура IF LocalResponse = RequestCompleted определяет, сделал ли пользователь выбор в вызванной процедуре и оператор %ControlUse = %LookupField присваивает выбранное значение USE переменной диалогового элемента.

Оператор #IF(%ControlEvent='Accepted' определяет, когда текстовый шаблон применяется для события EVENT:Accepted и добавляет вариант ELSE, чтобы выбрать (SELECT) снова диалоговый элемент и вернуться (CYCLE) обратно в начало ACCEPT цикла.

Оператор END завершает структуру IF LocalResponse = RequestCompleted. Структура #IF(%ControlEvent='Selected') генерирует оператор SELECT для диалогового элемента при генерации текста обработки события EVENT:Selected.

Оператор END завершает структуру IF ERRORCODE(). Очевидно, что если в операторе GET не было ошибок, то данные правильны и больше не нужен никакой текст.

Control Template: DOSFileLookup

Диалоговый шаблон DOSFileLookup размещает кнопку с многоточием (...), которая открывает стандартный диалог Windows Open File. Можно указать маски файлов и переменную для хранения результата выбора.

```
#CONTROL(DOSFileLookup,'Lookup a DOS file name'),WINDOW,MULTI
    CONTROLS
        BUTTON('...'),AT(,12,12),USE(?LookupFile)
    END
#BOXED('DOS File Lookup Prompts')
#PROMPT('&File Dialog Header:',@S60), %|
    %DOSFileDialogHeader,REQ,DEFAULT('Choose File')
#PROMPT('&DOS FileName Variable:',FIELD),%DOSFileField,REQ
#PROMPT('D&efault Directory:',@S80),%DOSInitialDirectory
#PROMPT('&Return to original directory when done.',CHECK), %|
    %ReturnToOriginalDir,AT(10)
#PROMPT('&Use a variable to specify the file mask(s).',CHECK), %|
    %DOSVariableMask,AT(10)
#ENABLE(%DOSVariableMask)
    #PROMPT('Vari&able Mask Value:',FIELD),%DOSVariableMaskValue
#ENDENABLE
```



```

#ENABLE(NOT %DOSVariableMask)
#PROMPT('F&ile Mask Description:',@S40), %|
    %DOSMaskDesc,REQ,DEFAULT('All Files')
#PROMPT('Fi&le Mask',@S50),%DOSMask,REQ,DEFAULT('*. *')
#BUTTON('More Fil&e Masks') %|
    ,MULTI(%DOSMoreMasks,%DOSMoreMaskDesc&'-'&%DOSMoreMask)
#PROMPT('File Mask Description:',@S40),%DOSMoreMaskDesc,REQ
#PROMPT('File Mask',@S50),%DOSMoreMask,REQ
#ENDBUTTON
#ENDENABLE
#ENDBOXED
#LOCALDATA
DOSDialogHeader      CSTRING(40)
DOSExtParameter      CSTRING(250)
DOSTargetVariable    CSTRING(80)
#ENDLOCALDATA
#ATSTART
#DECLARE(%DOSExtensionParameter)
#DECLARE(%DOSLookupControl)
#FOR(%Control),WHERE(%ControlInstance = %ActiveTemplateInstance)
    #SET(%DOSLookupControl,%Control)
#ENDFOR
#IF(NOT %DOSVariableMask)
    #SET(%DOSExtensionParameter,%DOSMaskDesc & '|' & %DOSMask)
    #FOR(%DOSMoreMasks)
        #SET(%DOSExtensionParameter,%DOSExtensionParameter %|
            & '|' & %DOSMoreMaskDesc & '|' & %DOSMoreMask)
    #ENDFOR
#END
#ENDAT
#AT(%ControlEventHandling,%DOSLookupControl,'Accepted')
IF NOT %DOSFileField
#INSERT(%StandardValueAssignment,'DOSTargetVariable', %|
    %DOSInitialDirectory)
ELSE
    DOSTargetVariable = %DOSFileField
END
#INSERT(%StandardValueAssignment,'DOSDialogHeader', %|
    %DOSFileDialogHeader)
#IF(%DOSVariableMask)
DOSExtParameter = %DOSVariableMaskValue
#ELSE
DOSExtParameter = '%DOSExtensionParameter'

```

```
#ENDIF
#IF(%ReturnToOriginalDir)
IF FILEDIALOG(DOSDialogHeader,DOSTargetVariable,DOSExtParameter, |
    FILE:KeepDIR)
#ELSE
IF FILEDIALOG(DOSDialogHeader,DOSTargetVariable,DOSExtParameter,0)
#ENDIF
    %DOSFileField = DOSTargetVariable
    DO RefreshWindow
END
#ENDAT
```

Он начинается, как и должен начинаться любой диалоговый шаблон, с оператора **#CONTROL**. Атрибут **WINDOW** позволяет размещать его на окне, но не на документе. Атрибут **MULTI** указывает, что шаблон может размещаться в окне несколько раз. Секция **CONTROLS** предопределяет оконный диалоговый элемент **BUTTON**.

Структура **#BOXED** размещает рамку вокруг всех приглашений, которые выводятся на страничке **Action** этого диалогового шаблона. Первый **#PROMPT** запрашивает текст заголовка для диалога **Open File**, а следующий - имя переменной где будет храниться имя выбранного файла. Третий позволяет явно задать каталог в котором откроется диалог **Open File**.

Четвертый **#PROMPT** - это переключатель, который указывает, должен ли производиться возврат в каталог, где был открыт диалог **Open File**. Пятый **#PROMPT** - это переключатель, определяющий, будут ли маски файлов диалога **Open File** указаны явно, или будет использоваться переменная для формирования их в период исполнения. При отметке первый **#ENABLE** активирует **Variable Mask Value #PROMPT** для ввода имени используемой во время исполнения переменной. При отсутствии отметки второй **#ENABLE** активирует свой набор приглашений для ввода каждой маски, которая будет передана в диалог **Open File**.

Секция **#LOCALDATA** объявляет три локальных переменных, который автоматически генерируются как часть процедуры. Эти локальные переменные используются только в тексте, который генерируется этим диалоговым шаблоном, в качестве фактических параметров функции **FILEDIALOG**.

Оператор **#ATSTART** начинает секцию шаблонного текста, который выполняется перед генерацией любого текста процедуры. Это означает, что здесь может только проводиться только инициализация пользовательских символов шаблона и выполняться необходимые предустановки для генерации в диалоговым шаблоном надлежащего текста процедуры. В этой секции не генерируется никакой исходный текст. Операторы **#DECLARE** объявляют два символа, которые используются только во время генерации текста этим диалоговым

шаблоном.

Оператор `#FOR(%Control),WHERE(%ControlInstance=%ActiveTemplateInstance)` выполняет `#SET` в теле цикла только для того единственного диалогового элемента, который был размещен этим шаблоном. Последующий оператор `#SET` помещает метку поля этого элемента в `%DOSLookupControl`.

Структура `#IF` проверяет, отметил ли программист переключатель “Use a Variable to specify the file mask(s) box”, и если нет, формирует заданные программистом маски файлов для передачи в функцию `FILEDIALOG`. Оператор `#ENDAT` завершает секцию `#AT`.

Следующий `#AT` генерирует Clarion-текст в точку обработки события Accepted для размещенного этим шаблоном элемента для выполнения поиска файла. Структура `IF NOT %DOSFileField` определяет, совершил ли пользователь выбор файла.

Если нет, в `DOSTargetVariable` записывается начальный каталог. Если пользователь совершил выбор, вариант `ELSE` запоминает результат выбора в качестве начальной точки для следующего раза.

Оператор `#INSERT` строит операторы присваивания для инициализации заголовка диалога Open File. Затем `#IF(%DOSVariableMask)` условно генерирует присваивание для инициализации передаваемых в диалог Open File масок файлов.

Оператор `#IF (%ReturnToOrigianlDir)` генерирует надлежащую структуру `IF FILEDIALOG` которая обеспечивает или не обеспечивает возврат в исходный каталог после выбора файла. (На самом деле флажок, который устанавливается привключенном `%ReturnToOrigianlDir`, запрещает пользователю изменять каталог во время выбора файла. Прим. перев.) Если пользователь выберет файл в диалоге Open File, имя файла записывается в переменную, которую программист указал в приглашении “DOS FileName Variable”, а оператор `DO RefreshWindow` обеспечивает обновление данных на экране. Оператор `#ENDAT` завершает секцию `#AT`.

Распределенный шаблон: DateTimeDisplay

Распределенный шаблон `DateTimeDisplay` выводит дату и/или время либо в выводном диалоговом элементе `STRING` либо в зоне строки состояния. Конечно, в окне должна быть объявлена строка состояния.

```
#EXTENSION(DateTimeDisplay,'Display the date and/or time in the current  
window') %|  
      ,HLP('~TPLExtensionDateTimeDisplay'),PROCEDURE
```

```

#BUTTON('Date and Time Display'),AT(10,,180)
#BOXED('Date Display...')
  #PROMPT('Display the current day/date in the window',CHECK) %|
    ,%DisplayDate,DEFAULT(0),AT(10,,150)
    #ENABLE(%DisplayDate)
    #PROMPT('Date Picture:',DROP('October 31, 1959|OCT 31,1959|10/31/59|
%|
    10/31/1959|31 OCT 59|31 OCT 1959|31/10/59| %|
    31/10/1959|Other')),%DatePicture %|
    ,DEFAULT('October 31, 1959')
    #ENABLE(%DatePicture = 'Other')
    #PROMPT('Other Date Picture:',@S20),%OtherDatePicture,REQ
  #ENDENABLE
    #PROMPT('Show the day of the week before the
date',CHECK),%ShowDayOfWeek %|
    ,DEFAULT(1),AT(10,,150)
    #PROMPT('&Location of Date Display:',DROP('Control|Status Bar')) %|
    ,%DateDisplayLocation
    #ENABLE(%DateDisplayLocation='Status Bar')
    #PROMPT('Status Bar
Section:',@n1),%DateStatusSection,REQ,DEFAULT(1)
  #ENDENABLE
  #ENABLE(%DateDisplayLocation='Control')
  #PROMPT('Date Display Control:',CONTROL),%DateControl,REQ
  #ENDENABLE
#ENDENABLE
#ENDBOXED
#BOXED('Time Display...')
  #PROMPT('Display the current time in the
window',CHECK),%DisplayTime %|
    ,DEFAULT(0),AT(10,,150)
    #ENABLE(%DisplayTime)
    #PROMPT('Time Picture:',DROP('5:30PM|5:30:00PM|17:30|17:30:00| %|
    1730|173000|Other')),%TimePicture %|
    ,DEFAULT('5:30PM')
    #ENABLE(%TimePicture = 'Other')
    #PROMPT('Other Time Picture:',@S20),%OtherTimePicture,REQ
  #ENDENABLE
  #PROMPT('&Location of Time Display:',DROP('Control|Status Bar')) %|
    ,%TimeDisplayLocation
  #ENABLE(%TimeDisplayLocation='Status Bar')
  #PROMPT('Status Bar
Section:',@n1),%TimeStatusSection,REQ,DEFAULT(2)

```

```
#ENDENABLE
#ENABLE(%TimeDisplayLocation='Control')
    #PROMPT('Time Display Control:',CONTROL),%TimeControl,REQ
#ENDENABLE
#ENDENABLE
#ENDBOXED
#ENDBUTTON
#ATSTART
    #DECLARE(%TimerEventGenerated)
    #IF(%DisplayDate)
        #DECLARE(%DateUsePicture)
        #CASE(%DatePicture)
            #OF('10/31/59')
                #SET(%DateUsePicture,'@D1')
            #OF('10/31/1959')
                #SET(%DateUsePicture,'@D2')
            #OF('OCT 31,1959')
                #SET(%DateUsePicture,'@D3')
            #OF('October 31, 1959')
                #SET(%DateUsePicture,'@D4')
            #OF('31/10/59')
                #SET(%DateUsePicture,'@D5')
            #OF('31/10/1959')
                #SET(%DateUsePicture,'@D6')
            #OF('31 OCT 59')
                #SET(%DateUsePicture,'@D7')
            #OF('31 OCT 1959')
                #SET(%DateUsePicture,'@D8')
            #OF('Other')
                #SET(%DateUsePicture,%OtherDatePicture)
        #ENDCASE
    #ENDIF
    #IF(%DisplayTime)
        #DECLARE(%TimeUsePicture)
        #CASE(%TimePicture)
            #OF('17:30')
                #SET(%TimeUsePicture,'@T1')
            #OF('1730')
                #SET(%TimeUsePicture,'@T2')
            #OF('5:30PM')
                #SET(%TimeUsePicture,'@T3')
            #OF('17:30:00')
                #SET(%TimeUsePicture,'@T4')
```

```

    #OF('173000')
        #SET(%TimeUsePicture,'@T5')
    #OF('5:30:00PM')
        #SET(%TimeUsePicture,'@T6')
    #OF('Other')
        #SET(%TimeUsePicture,%OtherTimePicture)
    #ENDCASE
#ENDIF
#ENDAT
#AT(%DataSectionBeforeWindow)
    #IF(%DisplayDate AND %ShowDayOfWeek)
DisplayDayString STRING('Sunday Monday Tuesday WednesdayThursday %|
                        Friday Saturday ')
DisplayDayText   STRING(9),DIM(7),OVER(DisplayDayString)
    #ENDIF
#ENDAT
#AT(%BeforeAccept)
    #IF(%DisplayTime OR %DisplayDate)
IF NOT INRANGE(%Window{Prop:Timer},1,100)
    %Window{Prop:Timer} = 100
END
#INSERT(%DateTimeDisplayCode)
    #ENDIF
#ENDAT
#AT(%WindowEventHandling,'Timer')
    #SET(%TimerEventGenerated,%True)
    #IF(%DisplayDate OR %DisplayTime)
#INSERT(%DateTimeDisplayCode)
    #ENDIF
#ENDAT
#AT(%WindowOtherEventHandling)
    #IF(%DisplayDate OR %DisplayTime)
        #IF(NOT %TimerEventGenerated)
IF EVENT() = EVENT:Timer
    #INSERT(%DateTimeDisplayCode)
END
        #ENDIF
    #ENDIF
#ENDAT

```

Распределенный шаблон начинается с оператора #EXTENSION. Атрибут PROCEDURE указывает, что распределенный шаблон доступен только на уровне процедуры, а не на

глобальном уровне приложения.

Структура `#BUTTON` создает новую страницу для всех элементов ввода для этого распределенный шаблона. Этот ввод запрашивает программиста о формате вывода даты и /или времени и где их нужно показывать: в диалоговом элементе или в строке состояния.

Оператор `#ATSTART` начинает секцию шаблона, которая выполняется перед генерацией любого текста этой процедуры. Это означает, что она предназначена только для инициализации определяемых шаблонных символов, и выполняет всю необходимую подготовку для генерации в процедуре правильного текста диалогового шаблона. Эта секция не генерирует исходный текст.

Оператор `#DECLARE(%TimerEventGenerated)` объявляет символ, который используется только в этом шаблоне. Он используется для пометки, что в процедуре генерируется вариант `OF EVENT:Timer`.

Структура `#IF(%DisplayDate)` подготавливает вывод даты, объявляя символ, содержащий выбранный программистом формат. Структура `#CASE` присваивает выбранный вариант символу `%DateUsePicture`. Структура `#IF(%DisplayTime)` подготавливает вывод времени, объявляя символ, содержащий выбранный программистом формат. Структура `#CASE` присваивает выбранный вариант символу `%TimeUsePicture`. Оператор `#ENDAT` завершает секцию `#ATSTART`.

Следующий оператор `#AT` размещает текст в точке вставки, которая расположена непосредственно перед структурой `window` в области данных. Структура `#IF(%DisplayDate AND %ShowDayOfWeek)` создает объявления двух локальных переменных процедуры, если программист заказал вывод даты вместе с днем недели.

Следующий оператор `#AT` размещает текст в точке вставки, которая расположена непосредственно перед циклом `ACCEPT`. Структура `#IF(%DisplayTime OR %DisplayDate)` генерирует текст, который обеспечивает установку атрибута `TIMER` в структуре `window`. Структура `IF NOT INRANGE(%Window{Prop:Timer},1,100)` определяет отсутствие атрибута, затем `%Window{Prop:Timer} = 100` устанавливает его равным одной секунде. Оператор `#INSERT(%DateTimeDisplayCode)` добавляет текст, который обновляет окно.

Следующая структура `#AT` размещает текст в точке вставки для события `EVENT:Timer`. Эта точка вставки появляется только, если программист указал в окне атрибут `TIMER`. Поэтому оператор `#SET(%TimerEventGenerated,%True)` сигнализирует, что текст был размещен в этой точке вставки. Структура `#IF(%DisplayTime OR %DisplayDate)` обеспечивает, что оператор `#INSERT(%DateTimeDisplayCode)` сгенерирует текст, который обновит окно каждый раз при обработке события `EVENT:Timer`.

Следующая структура #AT размещает текст в точке вставки “Other Window Event Handling”. Это вариант ELSE структуры CASE EVENT(), обрабатывающий независимые от поля события. Структура #IF(NOT %TimerEventGenerated) определяет, что предыдущая секция #AT не разместила текст, потому что программист не указал для окна атрибут TIMER. Поэтому необходима структура IF EVENT() = EVENT:Timer для текста, который обновляет окно при возникновении события EVENT:Timer.

%DateTimeDisplayCode #GROUP

Эта #GROUP генерирует текст для непосредственного вывода даты и/или времени.

```
#GROUP(%DateTimeDisplayCode)
  #IF(%DisplayDate)
    #IF(%ShowDayOfWeek)
      #CASE(%DateDisplayLocation)
        #OF('Control')
          %DateControl{Prop:Text} = CLIP(DisplayDayText[(TODAY())%%7)+1]) & ', ' & %|
                                FORMAT(TODAY(),%DateUsePicture)
          DISPLAY(%DateControl)
        #ELSE
          %Window{Prop:StatusText,%DateStatusSection} = CLIP(DisplayDayText[( %|
                                TODAY())%%7)+1]) & ', ' & FORMAT(TODAY(),%DateUsePicture)
          #ENDCASE
        #ELSE
          #CASE(%DateDisplayLocation)
            #OF('Control')
              %DateControl{Prop:Text} = FORMAT(TODAY(),%DateUsePicture)
              DISPLAY(%DateControl)
            #ELSE
              %Window{Prop:StatusText,%DateStatusSection} =
FORMAT(TODAY(),%DateUsePicture)
          #ENDCASE
        #ENDIF
      #ENDIF
    #IF(%DisplayTime)
      #CASE(%TimeDisplayLocation)
        #OF('Control')
          %TimeControl{Prop:Text} = FORMAT(CLOCK(),%TimeUsePicture)
          DISPLAY(%DateControl)
        #ELSE
          %Window{Prop:StatusText,%TimeStatusSection} =
FORMAT(CLOCK(),%TimeUsePicture)
```



```
#ENDCASE  
#ENDIF
```

Структура `#IF(%DisplayDate)` генерирует текст для вывода даты. Структура `#IF(%ShowDayOfWeek)` определяет, выбрал ли программист вывод дня недели вместе с датой, затем `#CASE(%DateDisplayLocation)` генерирует в варианте `#OF('Control')` текст для вывода в выводном диалоговом элементе `STRING`.

Оператор присваивания сцепляет день недели (из выражения `DisplayDayText[(TODAY() %% 7)+1]`) с форматированной датой (из выражения `FORMAT(TODAY(),%DateUsePicture)`) в свойство `STRING(text)` (свойство `%DateControl{Prop:Text}`). Символы `%%` генерируют единственный символ `%` (операция деления по модулю) в выражении `TODAY() % 7 + 1`, чтобы получить день недели в массиве `DisplayDayText`. Вариант `#ELSE` структуры `#CASE(%DateDisplayLocation)` записывает тоже самое выражение в зону строки состояния, которую выбрал программист для вывода даты.

ыВариант `#ELSE` структуры `#IF(%ShowDayOfWeek)` выполняет те же самые присваивания без дня недели. Структура `#IF(%DisplayTime)` выполняет аналогичные присваивания отформатированного времени либо в выводной диалоговый элемент `STRING`, либо в строку состояния.

Содержание

Глава 1 Введение в язык шаблонов	1
Обзор языка шаблонов	2
Что такое шаблоны	2
Виды шаблонов	4
Что делают шаблоны	4
Препроцессирование и генерация исходного текста	5
Точки вставки	6
Шаблонные элементы ввода	7
Взаимодействие со Словарем Данных	8
Структура шаблона	9
Формат исходного текста шаблона	9
Файл регистра шаблонов	10
Настройка стандартных шаблонов	11
Добавление новых наборов шаблонов	12
Глава 2 Устройство шаблона	15
Секции шаблона	16
#TEMPLATE (начало набора шаблонов)	16
#APPLICATION (секция управления генерацией текста)	16
#PROGRAM (область глобалов)	19
#MODULE (область модуля)	20
#PROCEDURE (начало процедурного шаблона)	20
#GROUP (повторно используемая группа операторов)	21
#UTILITY (секция утилит)	23
#CODE (определение текстового шаблона)	24
#CONTROL (определение диалогового шаблона)	26
#EXTENSION (определение распределенного шаблона)	30
Точки вставки	32
#EMBED (задание точек вставки исходного текста)	32
#AT (вставка текста в точку вставки)	34
#ATSTART (инициализация шаблона)	35
#ATEND (завершение шаблона)	36
#EMPTYEMBED (генерация комментария в пустой точке вставки)	36
#POSTEMBED (генерация комментария в конце точки вставки)	37
#PREEMBED (генерация комментария в конце точки вставки)	38
Поддержка целостности шаблона	38
#WHERE (указание доступности #CODE в точке вставки)	38

#RESTRICT (ограничения на использование секции)	39
#ACCEPT (разрешение на использование секции)	40
#REJECT (запрет на использование секции)	41

Глава 3 Стандартные структуры и шаблонные переменные .. 43

Стандартные данные и тексты	44
#WINDOWS (стандартная оконная структура)	44
#REPORTS (стандартная структура документа)	45
#LOCALDATA (стандартные объявления локальных данных)	46
#GLOBALDATA (стандартные объявления глобальных данных)	46
#DEFAULT (стандартная начальная процедура)	47
Операторы управления символами	48
#DECLARE (объявление определяемого символа)	48
#ALIAS (обращение к символу из другого экземпляра)	50
#ADD (добавление к многозначному символу)	50
#DELETE (удаление экземпляра многозначного символа)	51
#DELETEALL (удаление экземпляров многозначного символа)	52
#PURGE (удаление экземпляров однозначного или многозначного символа)	52
#CLEAR (очистка однозначного символа)	53
#FREE (очистка многозначного символа)	53
#FIX (фиксация многозначного символа)	54
#FIND (“суперфиксация” многозначных символов)	55
#SELECT (фиксация многозначного символа)	56
#SET (присваивание значения определяемому символу)	56
#UNFIX (расфиксация многозначного символа)	57
Атрибуты оператора #DECLARE	58
UNIQUE (запрет повторений)	58
SAVE (сохранение значений символа между генерациями)	58

Глава 4 Ввод программиста..... 59

Операторы вывода и оформления.	59
Операторы ввода и проверки	60
#PROMPT (приглашение к вводу)	60
#VALIDATE (проверка ввода)	63
#ENABLE (включение/выключение элементов диалога)	64
#BUTTON (вызов новой страницы диалога)	64
#FIELD (ввод для диалоговых элементов окна)	67
#PREPARE (подготовить символы приглашений)	68
Варианты type в #PROMPT	69
CHECK (переключатель)	69
COMPONENT (список полей ключа)	69
CONTROL (список полей окна)	69

<i>DROP</i> (выпадающий список)	70
<i>EMBED</i> (ввод текста в точку вставки)	70
<i>EMBEDBUTTON</i> (ввод текста в точку вставки)	71
<i>FIELD</i> (список полей данных)	72
<i>FILE</i> (список файлов)	72
<i>FORMAT</i> (вызов форматера списковой структуры)	73
<i>FROM</i> (список значений символа)	73
<i>KEY</i> (список ключей)	73
<i>KEYCODE</i> (список кодов клавиш)	74
<i>OPENDIALOG</i> (вызвать диалог Open File)	74
<i>OPTION</i> (вывод радиокнопок)	74
<i>PICTURE</i> (вызов форматера шаблонов ввода)	75
<i>PROCEDURE</i> (добавление процедуры к дереву процедур)	75
<i>RADIO</i> (единичная радиокнопка)	75
<i>SAVEDIALOG</i> (вызвать диалог Save File)	76
<i>SPIN</i> (элемент диалога с “прокруткой”)	76
Операторы вывода и оформления.	77
#BOXED (группа элементов диалога)	77
#DISPLAY (вывод)	78
#IMAGE (вывести графическое изображение)	78
#SHEET (объявляет группу диалоговых элементов #TAB)	78
#TAB (объявляет страницу диалогового элемента #SHEET)	79
Глава 5 Управление генерацией текста	83
Операторы управления логикой	84
#FOR (многократная генерация текста)	84
#IF (условная генерация текста)	85
#LOOP (итеративная генерация текста)	86
#CASE (структура условного выполнения)	88
#INSERT (вставить текст из #GROUP)	89
#BREAK (выход из цикла)	90
#CYCLE (переход в начало цикла)	90
#RETURN (возврат из #GROUP)	91
#GENERATE (генерация секции исходного текста)	91
#ABORT (прекращение генерации)	92
Операторы управления файлами	92
#CREATE (создание файла исходного текста)	92
#OPEN (открыть файл исходного текста)	93
#CLOSE (закрыть файл исходного текста)	94
#READ (чтение одной строки из файла исходного текста)	95
#REDIRECT (изменить файл исходного текста)	95
#APPEND (добавить в конец файла исходного текста)	96

#REMOVE (удаление файла исходного текста)	97
#REPLACE (условная замена файла исходного текста)	98
#PRINT (печать файла исходного текста)	98
Операторы условной генерации	99
#SUSPEND (начать условный текст)	99
#RELEASE (подтверждение условной генерации текста)	100
#RESUME (граница условной генерации текста)	101
#? (условная строка условного текста)	102

Глава 6 Прочие операторы 105

Прочие операторы.....	106
#! (комментарии в тексте шаблона)	106
#< (выровненные комментарии целевого языка)	106
#CLASS (определение класса формулы)	107
#COMMENT (определение колонки комментария)	107
#ERROR (вывод ошибок генерации исходного текста)	108
#EXPORT (вывести символ в текст)	108
#HELP (определение файла справочника по шаблонам)	109
#INCLUDE (включение файла шаблона)	109
#IMPORT (импорт .APP из сценария)	110
#MESSAGE (вывод сообщения о генерации исходного текста)	110
#PROTOTYPE (прототип процедуры)	111
#PROJECT (добавление файла к проекту)	112

Встроенные шаблонные функции	113
EXTRACT (атрибут)	113
EXISTS (существование точки вставки)	113
FILEEXISTS (существование файла)	114
INLIST (существование элемента списка)	114
INSTANCE (текущий номер экземпляра символа)	115
ITEMS (количество экземпляров многозначного символа)	115
LINKNAME (возвращает преобразованное имя процедуры)	115
QUOTE (замена в строке специальных символов)	116
REPLACE (замена атрибута)	116
SEPARATOR (положение разделителя в строке атрибутов)	117

Глава 7 Символы шаблонов 119

Обзор символов	120
Символы расширения	120
Обзор иерархии символов	121
Встроенные символы	122
Символы, зависящие от %Application	122
Символы, зависящие от %File	123

Символы, зависящие от %ViewFiles	125
Символы, зависящие от %Field	126
Символы, зависящие от %Key	128
Символы, зависящие от %Relation	129
Символы, зависящие от %Module	130
Символы, зависящие от %Procedure	130
Символы оконных диалоговых элементов	132
Символы элементов данных документа	134
Символы формул	135
Символы файловой схемы	136
Символы файлового драйвера	136
Прочие символы	137

Глава 8 Аннотированные примеры 139

Процедурный шаблон: Window	140
%StandardWindowCode #GROUP	142
%StandardWindowHandling #GROUP	149
"%StandardAcceptHandling #GROUP	152
%StandardControlHandling #GROUP	153
Текстовый шаблон: ControlValueValidation	156
%CodeTPLValidationCode #GROUP	158
Control Template: DOSFileLookup	160
Распределенный шаблон: DateTimeDisplay	163
%DateTimeDisplayCode #GROUP	168